IATO, IA64 Toolkit

Tools and Library Reference

Amaury Darsch - Pierre Villalon - Andre Seznec

Contents

Part I

ISA Library

Chapter 1

ISA Library Compound Index

1.1 IATO ISA LIBRARY Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

ISA Library Reference

2.1 iato::Aexecute Class Reference

`#include <Aexecute.hpp>`

Inherits **iato::Executable**.

Inherited by **iato::Iexecute**, and **iato::Mexecute**.

Public Member Functions

- **Result exec** (const **Instr** &inst, const **Operand** &oprd) const

2.1.1 Detailed Description

The **Aexecute**(p. 5) is the A execution unit. This unit is shared by both M and I units. Normally, this class should not be used directly.

2.1.2 Member Function Documentation

2.1.2.1 Result iato::Aexecute::exec (const Instr & *inst*, const Operand & *oprd*) const [virtual]

execute this instruction with its operand

Parameters:

inst the instruction to execute

oprd the operand object

Implements **iato::Executable** (p. 27).

Reimplemented in **iato::Iexecute** (p. 34), and **iato::Mexecute** (p. 62).

The documentation for this class was generated from the following file:

- Aexecute.hpp

2.2 iato::Alat Class Reference

`#include <Alat.hpp>`

Inherits **iato::Resource**.

Public Member Functions

- **Alat** (void)
  create a default alat
- **Alat** (**Ctx** *ctx)
- **Alat** (**Ctx** *ctx, const string &name)
- ~**Alat** (void)
  destroy this alat
- void **reset** (void)
  reset this alat
- void **report** (void) const
  report this alat
- bool **isfull** (void) const
  true if the alat is full
- bool **isvalid** (const t_word ind) const
  true if the line is valid
- virtual t_word **gettag** (const **Rid** reg) const
- void **add** (const t_octa addr, const **Rid** reg, const t_byte size)
  add a new entry in the alat
- void **memupd** (const t_octa addr, const t_byte size)
- void **remove** (const **Rid** reg)
- bool **check** (const **Rid** reg) const
- bool **load** (**Result** &resl, const long indx)
- void **check** (**Result** &resl, const long indx)

2.2.1 Detailed Description

The **Alat**(p. 6) class is an abstract class definition of the **Alat**(p. 6). It defines the basic interface for any **Alat**(p. 6), a default implementation for each interface is given.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 iato::Alat::Alat (Ctx * *ctx*)

create an alat with a context

Parameters:

ctx the current context

2.2.2.2 iato::Alat::Alat (Ctx * *ctx*, const string & *name*)

create an alat with a context and name

Parameters:

ctx the current context

name the resource name

2.2.3 Member Function Documentation

2.2.3.1 void iato::Alat::check (Result & *resl*, const long *indx*)

check and update result

Parameters:

resl the result which contains the check

indx the check index in the result

2.2.3.2 bool iato::Alat::check (const Rid *reg*) const

Parameters:

reg the **Rid**(p. 102) to look for

Returns:

true if an entry matches

2.2.3.3 virtual t_word iato::Alat::gettag (const Rid *reg*) const [virtual]

get a tag according to register type and number

Parameters:

reg the regiter rid

Returns:

the tag value

2.2.3.4 bool iato::Alat::load (Result & *resl*, const long *indx*)

determine if the load have to be done and update alat state

Parameters:

resl the result which contains the load

indx the load index in the result

Returns:

true if the load have to be done

2.2.3.5 void iato::Alat::memupd (const t_octa *addr*, const t_byte *size*)

notify to alat that a memory address have been changed

Parameters:

addr the memory address write accessed

size the size of the access

2.2.3.6 void iato::Alat::remove (const Rid *reg*)

remove an entry from the alat

Parameters:

reg the **Rid**(p. 102) to remove

The documentation for this class was generated from the following file:

- Alat.hpp

2.3 iato::Bexecute Class Reference

`#include <Bexecute.hpp>`

Inherits **iato::Executable**.

Public Member Functions

- **Result exec** (const **Instr** &inst, const **Operand** &oprd) const

2.3.1 Detailed Description

The **Bexecute**(p. 9) class is the B execution unit. The class is responsible for all branch operations.

2.3.2 Member Function Documentation

2.3.2.1 Result iato::Bexecute::exec (const Instr & *inst*, const Operand & *oprd*) const [virtual]

execute this instruction with its operand

Parameters:

inst the instruction to execute

oprd the operand object

Implements **iato::Executable** (p. 27).

The documentation for this class was generated from the following file:

- Bexecute.hpp

2.4 iato::Bundle Class Reference

`#include <Bundle.hpp>`

Public Member Functions

- **Bundle** (void)
  create a default bundle
- ∼**Bundle** (void)
  destroy this bundle
- **Bundle** (const **Bundle** &that)
- **Bundle** & **operator=** (const **Bundle** &that)
- void **reset** (void)
  reset this bundle
- bool **isvalid** (void) const
  true if the bundle is valid
- bool **isbr** (const long slot) const
  true if the bundle slot is a branch
- void **sethist** (const t_octa hist)
- t_octa **gethist** (void) const
  the predictor history
- void **setbip** (const t_octa ip)
- t_octa **getbip** (void) const
  the bundle instruction pointer
- void **setsip** (const t_octa sip, const long ssl)
- void **setsip** (const t_octa sip, const long ssl, const t_octa hist)
- t_octa **getsip** (void) const
  the bundle speculative ip
- void **setvsb** (const long slot, const bool flag)
- bool **getvsb** (const long slot) const
  the valid slot bit by index
- long **length** (void) const
  the bundle length in bytes
- void **push** (const t_byte byte)
- t_byte **get** (const long index) const
  index the buffer index
- t_byte **gettmpl** (void) const
  the bundle template code

- t_octa **getslot** (const long slot) const
- **Instr getinstr** (const long slot) const

  get an instruction by slot index

- string **tostring** (void) const

  a string representation of this bundle

- string **repr** (void) const

  a string representation of this bundle with stop bit

Static Public Member Functions

- string **tostrws** (const t_byte tmpl)
- string **tostrwo** (const t_byte tmpl)

Static Public Attributes

- const t_byte **BN_MxIxIx** = 0x00

  MII template.

- const t_byte **BN_MxIxIs** = 0x01

  MII| template.

- const t_byte **BN_MxIsIx** = 0x02

  MI|I template.

- const t_byte **BN_MxIsIs** = 0x03

  MI|I| template.

- const t_byte **BN_MxLxXx** = 0x04

  MLX template.

- const t_byte **BN_MxLxXs** = 0x05

  MLX| template.

- const t_byte **BN_MxMxIx** = 0x08

  MMI template.

- const t_byte **BN_MxMxIs** = 0x09

  MMI| template.

- const t_byte **BN_MsMxIx** = 0x0A

  M|MI template.

- const t_byte **BN_MsMxIs** = 0x0B

  M|MI| template.

- const t_byte **BN_MxFxIx** = 0x0C
  MFI template.
- const t_byte **BN_MxFxIs** = 0x0D
  MFI| template.
- const t_byte **BN_MxMxFx** = 0x0E
  MMF template.
- const t_byte **BN_MxMxFs** = 0x0F
  MMF| template.
- const t_byte **BN_MxIxBx** = 0x10
  MIB template.
- const t_byte **BN_MxIxBs** = 0x11
  MIB| template.
- const t_byte **BN_MxBxBx** = 0x12
  MBB template.
- const t_byte **BN_MxBxBs** = 0x13
  MBB| template.
- const t_byte **BN_BxBxBx** = 0x16
  BBB template.
- const t_byte **BN_BxBxBs** = 0x17
  BBB| template.
- const t_byte **BN_MxMxBx** = 0x18
  MMB template.
- const t_byte **BN_MxMxBs** = 0x19
  MMB| template.
- const t_byte **BN_MxFxBx** = 0x1C
  MFB template.
- const t_byte **BN_MxFxBs** = 0x1D
  MFB| template.
- const long **BN_MAXTPL** = 32
  max number of bundle templates

2.4.1 Detailed Description

The ”Bundle” class is a converter class that is loaded by bytes. Once loaded, the byte stream is decoded like a bundle. The bundle template, and the instruction can be accessed as well. A string representation of the template encoding is also available.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 iato::Bundle::Bundle (const Bundle & *that*)

copy construct this bundle

Parameters:
that the bundle to copy

2.4.3 Member Function Documentation

2.4.3.1 t_octa iato::Bundle::getslot (const long *slot*) const

get an instruction code by slot index

Parameters:
slot the instruction slot

2.4.3.2 Bundle& iato::Bundle::operator= (const Bundle & *that*)

assign a bundle to this one

Parameters:
that the bundle to assign

2.4.3.3 void iato::Bundle::push (const t_byte *byte*)

push a byte in the buffer

Parameters:
byte the byte to push

2.4.3.4 void iato::Bundle::setbip (const t_octa *ip*)

set the bundle instruction pointer

Parameters:
ip the instruction pointer

2.4.3.5 void iato::Bundle::sethist (const t_octa *hist*)

set the predictor history

Parameters:
hist the history to set

2.4.3.6 void iato::Bundle::setsip (const t_octa *sip*, const long *ssl*, const t_octa *hist*)

set the bundle speculative ip with the history

Parameters:

sip the speculative ip

ssl the speculative slot

hist the proedictor history

2.4.3.7 void iato::Bundle::setsip (const t_octa *sip*, const long *ssl*)

set the bundle speculative ip

Parameters:

sip the speculative ip

ssl the speculative slot

2.4.3.8 void iato::Bundle::setvsb (const long *slot*, const bool *flag*)

set the valid slot bit

Parameters:

slot the slot index

flag the bit value

2.4.3.9 string iato::Bundle::tostrwo (const t_byte *tmpl*) [static]

return the string representation of a template without stop bits

Parameters:

tmpl the template value

2.4.3.10 string iato::Bundle::tostrws (const t_byte *tmpl*) [static]

return the string representation of a template with stop bits

Parameters:

tmpl the template value

The documentation for this class was generated from the following file:

- Bundle.hpp

2.5 iato::Cfm Class Reference

`#include <Cfm.hpp>`

Public Types

- enum **t_field**

  the cfm fields

Public Member Functions

- **Cfm** (void)

  create a default cfm

- **Cfm** (const t_octa value)
- **Cfm** (const **Cfm** &that)
- **Cfm** & **operator=** (const **Cfm** &that)
- bool **operator==** (const **Cfm** &cfm) const

  true if two cfm are equals

- bool **operator==** (const t_octa val) const

  true if the cfm match an octa

- bool **operator!=** (const **Cfm** &cfm) const

  true if two cfm are not equal

- bool **operator!=** (const t_octa val) const

  true if the cfm do not match an octa

- void **reset** (void)

  reset this cfm

- void **setcfm** (const t_octa cfm)
- t_octa **getcfm** (void) const

  the resource cfm

- t_octa **rotate** (void)

  the updated rotated cfm

- void **setfld** (**t_field** fld, t_byte val)
- t_byte **getfld** (**t_field** fld) const

  the cfm field

- long **getrrb** (**t_field** fld) const

  the rrb field as a negative value

- void **setrrb** (**t_field** fld, const long val)
- void **clrrrb** (void)

clean the rrb

- void **alloc** (const t_byte sof, const t_byte sol, const t_byte sor)
- void **call** (const **Cfm** &cfm)

Static Public Attributes

- const long **GR_RRBM** = 128
  max value of rrb gr
- const long **FR_RRBM** = 128
  max value of rrb fr
- const long **PR_RRBM** = 64
  max value of rrb pr

2.5.1 Detailed Description

The **Cfm**(p. 15) class is the current frame marker class object. It holds all the field that are defined by the IA64 ISA. The cfm value is stored as a single field but specific method can be used to access a particular one.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 iato::Cfm::Cfm (const t_octa *value*)

create a cfm with a value

Parameters:
value the cfm value to set

2.5.2.2 iato::Cfm::Cfm (const Cfm & *that*)

copy construct this cfm

Parameters:
that the cfm to copy

2.5.3 Member Function Documentation

2.5.3.1 void iato::Cfm::alloc (const t_byte *sof*, const t_byte *sol*, const t_byte *sor*)

perform an alloc operation

Parameters:
sof the size of frame
sol the size of locals
sor the size of rotating

2.5.3.2 void iato::Cfm::call (const Cfm & *cfm*)

perform a call operation

Parameters:
cfm the cfm used for the call

2.5.3.3 Cfm& iato::Cfm::operator= (const Cfm & *that*)

assign a cfm to this one

Parameters:
that the cfm to assign

2.5.3.4 void iato::Cfm::setcfm (const t_octa *cfm*)

set the cfm value

Parameters:
cfm the cfm to set

2.5.3.5 void iato::Cfm::setfld (t_field *fld*, t_byte *val*)

set the cfm by field and value

Parameters:
fld the cfm field
val the cfm field value

2.5.3.6 void iato::Cfm::setrrb (t_field *fld*, const long *val*)

set the cfm rrb by field and value

Parameters:
fld the cfm field
val the cfm field value

The documentation for this class was generated from the following file:

- Cfm.hpp

2.6 iato::Checker Class Reference

`#include <Checker.hpp>`

Public Member Functions

- **Checker** (void)
  create a default checker
- **Checker** (**Ctx** *ctx)
- virtual ~**Checker** (void)
  destroy this checker
- virtual void **reset** (void)
  reset this checker
- virtual void **add** (const **Record** &rcd)
- virtual bool **check** (**Register** *rbk) const
  check all record against the register bank

2.6.1 Detailed Description

The **Checker**(p. 18) class is a simple record checker. The idea is to store a suite of record (typed register check) and compare them with a register bank. If a difference is found an error can be reported. The idea behind the checker is to automate the test suite process with simple verification.

2.6.2 Constructor & Destructor Documentation

2.6.2.1 iato::Checker::Checker (Ctx * *ctx*)

create a checker with a context

Parameters:
ctx the current context

2.6.3 Member Function Documentation

2.6.3.1 virtual void iato::Checker::add (const Record & *rcd*) [virtual]

add a record to the check vector

Parameters:
rcd the record to add

The documentation for this class was generated from the following file:

- Checker.hpp

2.7 iato::Ctx Class Reference

`#include <Ctx.hpp>`

Public Member Functions

- **Ctx** (void)
  create a new context
- virtual ~**Ctx** (void)
  destroy this context
- virtual void **reset** (void)
  reset this context
- void **parse** (const string &s)
- virtual void **parse** (const vector< string > &cprm)
- void **dump** (void) const
  dump this context to the standard output
- virtual void **update** (const t_arch arch)
- virtual void **setbool** (const string &what, const bool value)
- virtual bool **getbool** (const string &what) const
- virtual void **setlong** (const string &what, const long value)
- virtual long **getlong** (const string &what) const
- virtual void **setllong** (const string &what, const t_long value)
- virtual t_long **getllong** (const string &what) const
- virtual void **setreal** (const string &what, const long double value)
- virtual long double **getreal** (const string &what) const
- virtual void **setstr** (const string &what, const string &value)
- virtual string **getstr** (const string &what) const
- virtual long **getiwsz** (void) const
  the issue width size in bundles
- virtual long **getswsz** (void) const
  the issue width size in slots
- virtual long **getbwsz** (void) const
  the issue width size in bytes

Protected Attributes

- map< string, bool > **d_bmap**
  the boolean map
- map< string, long > **d_lmap**
  the long map

- map< string, t_long > **d_hmap**
  the 64 bits long map

- map< string, long double > **d_rmap**
  the real map

- map< string, string > **d_smap**
  the string map

2.7.1 Detailed Description

The **Ctx**(p. 19) class is a context class that is used to hold the architectural parameters. The default parameters for an architecture are defined at construction. They can be changed globally or individually. Most of the parameters can also be defined with a configuration file. The query is done by name.

2.7.2 Member Function Documentation

2.7.2.1 virtual bool iato::Ctx::getbool (const string & *what*) const [virtual]

return a boolean parameter value

Parameters:
what the parameter to query

2.7.2.2 virtual t_long iato::Ctx::getllong (const string & *what*) const [virtual]

return a long long parameter value

Parameters:
what the parameter to query

2.7.2.3 virtual long iato::Ctx::getlong (const string & *what*) const [virtual]

return a long parameter value

Parameters:
what the parameter to query

2.7.2.4 virtual long double iato::Ctx::getreal (const string & *what*) const [virtual]

return a real parameter value

Parameters:
what the parameter to query

2.7.2.5 virtual string iato::Ctx::getstr (const string & *what*) const [virtual]

return a string parameter value

Parameters:
what the parameter to query

2.7.2.6 virtual void iato::Ctx::parse (const vector< string > & *cprm*) [virtual]

update this context with a vector of parameters

Parameters:
cprm the vector of parameters

2.7.2.7 void iato::Ctx::parse (const string & *s*)

parse a string and update the context

Parameters:
s the string to parse

2.7.2.8 virtual void iato::Ctx::setbool (const string & *what*, const bool *value*) [virtual]

set a boolean parameter value

Parameters:
what the parameter to set
value the parameter value

2.7.2.9 virtual void iato::Ctx::setllong (const string & *what*, const t_long *value*) [virtual]

set a long long parameter value

Parameters:
what the parameter to set
value the parameter value

2.7.2.10 virtual void iato::Ctx::setlong (const string & *what*, const long *value*) [virtual]

set a long parameter value

Parameters:
what the parameter to set
value the parameter value

2.7.2.11 virtual void iato::Ctx::setreal (const string & *what*, const long double *value*) [virtual]

set a real parameter value

Parameters:

what the parameter to set

value the parameter value

2.7.2.12 virtual void iato::Ctx::setstr (const string & *what*, const string & *value*) [virtual]

set a string parameter value

Parameters:

what the parameter to set

value the parameter value

2.7.2.13 virtual void iato::Ctx::update (const t_arch *arch*) [virtual]

update this context with a particular architecture

Parameters:

arch the architecture used for the update

The documentation for this class was generated from the following file:

- Ctx.hpp

2.8 iato::Env Class Reference

`#include <Env.hpp>`

Public Member Functions

- **Env** (void)

  create a new environment

- virtual ~**Env** (void)

  destroy this environment

- virtual void **reset** (void)

  reset this environment

- virtual void **add** (**Resource** *res)
- virtual **Resource** * **get** (const string &name) const

  get a resource by name

- virtual void **report** (void) const

  report all resources

- virtual void **setstc** (**Stat** *stc)
- virtual void **settrc** (**Tracer** *tracer)

2.8.1 Detailed Description

The Environment class is an abstract class that defines an execution environment. Such environment is used to hold the global processor resources. The environment is by itself dependant on the processor architecture. For example, an out-of-order architecture might have a physical register file, a reorder buffer and other things.

2.8.2 Member Function Documentation

2.8.2.1 virtual void iato::Env::add (Resource * *res*) [virtual]

add a new resource to this environment

Parameters:

res the resource to add

2.8.2.2 virtual void iato::Env::setstc (Stat * *stc*) [virtual]

set the resource stat collector

Parameters:

stc the stat collector

2.8.2.3 virtual void iato::Env::settrc (Tracer * *tracer*) [virtual]

set the resource tracer to all resources

Parameters:

tracer the resource tracer to bind

The documentation for this class was generated from the following file:

- Env.hpp

2.9 iato::Exception Class Reference

`#include <Exception.hpp>`

Inherited by **iato::Interrupt**.

Public Member Functions

- **Exception** (void)

  create a default exception

- **Exception** (const string &type)
- **Exception** (const string &type, const string &reason)
- **Exception** (const **Exception** &that)
- virtual ~**Exception** (void)

  destroy this exception

- **Exception** & **operator=** (const **Exception** &that)
- virtual string **gettype** (void) const

  the exception type

- virtual string **getreason** (void) const

  the exception reason

- virtual void **setcycle** (const t_long cycle)
- virtual t_long **getcycle** (void) const

  the exception cycle

- virtual void **print** (void) const

  print the exception message

- virtual void **abort** (void) const

  print the exception and abort

Protected Attributes

- string **d_type**

  the exception type

- string **d_reason**

  the exception reason

- t_long **d_cycle**

  the exception cycle

2.9.1 Detailed Description

The **Exception**(p. 25) class is a basic class that is used exceptional runtime event, like error or priviledge violation. An exception has a type and a reason.

2.9.2 Constructor & Destructor Documentation

2.9.2.1 iato::Exception::Exception (const string & *type*)

create a new exception

Parameters:
type the exception type

2.9.2.2 iato::Exception::Exception (const string & *type*, const string & *reason*)

create a new exception

Parameters:
type the exception type
reason the exception reason

2.9.2.3 iato::Exception::Exception (const Exception & *that*)

copy construct this exception

Parameters:
that the exception to copy

2.9.3 Member Function Documentation

2.9.3.1 Exception& iato::Exception::operator= (const Exception & *that*)

assign an exception to this one

Parameters:
that the exception to assign

2.9.3.2 virtual void iato::Exception::setcycle (const t_long *cycle*) [virtual]

set the exception cycle

Parameters:
cycle the exception cycle to set

The documentation for this class was generated from the following file:

- Exception.hpp

2.10 iato::Executable Class Reference

`#include <Executable.hpp>`

Inherited by **iato::Aexecute**, **iato::Bexecute**, and **iato::Fexecute**.

Public Member Functions

- virtual **Result exec** (const **Instr** &inst, const **Operand** &oprd) const=0

2.10.1 Detailed Description

The **Executable**(p. 27) class is an abstract class for various execution unit. The class defines an exec method that uses an operand and produces a result. The instruction to execute is also used as an argument to the exec method.

2.10.2 Member Function Documentation

2.10.2.1 virtual Result iato::Executable::exec (const Instr & *inst*, const Operand & *oprd*) const `[pure virtual]`

execute this instruction with its operand

Parameters:

inst the instruction to execute

oprd the operand object

Implemented in **iato::Aexecute** (p. 5), **iato::Bexecute** (p. 9), **iato::Fexecute** (p. 28), **iato::Iexecute** (p. 34), and **iato::Mexecute** (p. 62).

The documentation for this class was generated from the following file:

- Executable.hpp

2.11 iato::Fexecute Class Reference

`#include <Fexecute.hpp>`

Inherits **iato::Executable**.

Public Member Functions

- **Result exec** (const **Instr** &inst, const **Operand** &oprd) const

2.11.1 Detailed Description

The **Fexecute**(p. 28) class is the F execution unit. The F unit is responsible to execute the floating point operations.

2.11.2 Member Function Documentation

2.11.2.1 Result iato::Fexecute::exec (const Instr & *inst*, const Operand & *oprd*) const `[virtual]`

execute this instruction with its operand

Parameters:

inst the instruction to execute

oprd the operand object

Implements **iato::Executable** (p. 27).

The documentation for this class was generated from the following file:

- Fexecute.hpp

2.12 iato::Filter Class Reference

`#include <Filter.hpp>`

Public Member Functions

- **Filter** (void)

  constructor

- virtual ∼**Filter** (void)

  destroy a filter

- **Filter** & **operator=** (const **Filter** &that)
- virtual void **reset** (void)

  reset this filter

- virtual void **setig** (const string &ig)
- virtual bool **check** (const **Record** &rcd) const
- virtual void **filter** (**Tracer::t_vrcd** ∗dst, const **Tracer::t_vrcd** ∗src) const

2.12.1 Detailed Description

The **Filter**(p. 29) class is a base class for record filtering. The base filter is an instruction group filter that can be used for statistical computation. At this level, the desired instruction group are added and a test method is provided to select a particular record as an active one or not.

2.12.2 Member Function Documentation

2.12.2.1 virtual bool iato::Filter::check (const Record & *rcd*) const [virtual]

check a record for selection

Parameters:

rcd the record to test

2.12.2.2 virtual void iato::Filter::filter (Tracer::t_vrcd ∗ *dst*, const Tracer::t_vrcd ∗ *src*) const [virtual]

filter a record vector

Parameters:

dst the destination record vector

src the source record vector

2.12.2.3 Filter& iato::Filter::operator= (const Filter & *that*)

assign a filter to this one

Parameters:
that the filter to assign

2.12.2.4 virtual void iato::Filter::setig (const string & *ig*) [virtual]

set the instruction group to trace

Parameters:
ig the instruction groups to filter

The documentation for this class was generated from the following file:

- Filter.hpp

2.13 iato::Fpsr Class Reference

`#include <Fpsr.hpp>`

Public Types

- enum **t_mfield**

  the fpsr field

- enum **t_field**

  trap fields and status fields

Public Member Functions

- **Fpsr** (void)

  create a default fpsr

- **Fpsr** (const t_octa value)
- **Fpsr** (const **Fpsr** &that)
- **Fpsr** & **operator=** (const **Fpsr** &that)
- void **reset** (void)

  reset this fpsr

- void **setfpsr** (const t_octa value)
- t_octa **getfpsr** (void) const

  get the current fpsr value

- void **setfld** (const **t_mfield** mfld, const **t_field** fld, const bool value)
- void **setfld** (const **t_mfield** mfld, const **t_field** fld, const t_byte value)
- bool **getbfld** (const **t_mfield** mfld, const **t_field** fld) const

  the fpsr bool field by mainfield, field

- t_byte **getbsfld** (const **t_mfield** mfld, const **t_field** fld) const

  the fpsr 2 bits field by mainfield, field

- void **convert** (const t_fpipc ipc, const **t_mfield** field, **t_real** &val)
- void **pconvert** (const **t_mfield** field, **t_real** &val1, **t_real** &val2)

2.13.1 Detailed Description

The **Fpsr**(p. 31) class is the floating point status register class object. It holds all the field that are defined by the IA64 ISA. The **Fpsr**(p. 31) value is stored as a single field but specific method can be used to access a particular one. It stands in the application register bank at ar40.

2.13.2 Constructor & Destructor Documentation

2.13.2.1 iato::Fpsr::Fpsr (const t_octa *value*)

create a fpsr with a value

Parameters:
value the fpsr value to set

2.13.2.2 iato::Fpsr::Fpsr (const Fpsr & *that*)

copy construct this fpsr

Parameters:
that the fpsr to copy

2.13.3 Member Function Documentation

2.13.3.1 void iato::Fpsr::convert (const t_fpipc *ipc*, const t_mfield *field*, t_real & *val*)

convert ia floating point register

Parameters:
ipc the instruction precision control completer

field the floating point status field format

val the real to convert

2.13.3.2 Fpsr& iato::Fpsr::operator= (const Fpsr & *that*)

assign a fpsr to this one

Parameters:
that the fpsr to assign

2.13.3.3 void iato::Fpsr::pconvert (const t_mfield *field*, t_real & *val1*, t_real & *val2*)

convert ia floating point register for parallel computing

Parameters:
field the floating point status field format

val1 the first real to convert

val2 the second real to convert

2.13.3.4 void iato::Fpsr::setfld (const t_mfield *mfld*, const t_field *fld*, const t_byte *value*)

set the fpsr by mainfield, field and value

Parameters:

mfld the main field

fld the field

value the value to set

2.13.3.5 void iato::Fpsr::setfld (const t_mfield *mfld*, const t_field *fld*, const bool *value*)

set the fpsr by mainfield, field and value

Parameters:

mfld the main field

fld the field

value the value to set

2.13.3.6 void iato::Fpsr::setfpsr (const t_octa *value*)

set the fpsr by value

Parameters:

value the value to set

The documentation for this class was generated from the following file:

- Fpsr.hpp

2.14 iato::Iexecute Class Reference

`#include <Iexecute.hpp>`

Inherits **iato::Aexecute**.

Public Member Functions

- **Result exec** (const **Instr** &inst, const **Operand** &oprd) const

2.14.1 Detailed Description

The **Iexecute**(p. 34) class is the I execution unit. The class is derived from the A execution unit.

2.14.2 Member Function Documentation

2.14.2.1 Result iato::Iexecute::exec (const Instr & *inst*, const Operand & *oprd*) const `[virtual]`

execute this instruction with its operand

Parameters:

inst the instruction to execute

oprd the operand object

Reimplemented from **iato::Aexecute** (p. 5).

The documentation for this class was generated from the following file:

- Iexecute.hpp

2.15 iato::Instr Class Reference

`#include <Instr.hpp>`

Public Member Functions

- **Instr** (void)
  create a default instruction
- **Instr** (t_unit unit, long slot, bool bstp, t_octa inst)
- **Instr** (t_unit unit, long slot, bool bstp, t_octa inst, t_octa extd)
- **Instr** (const **Instr** &that)
- **Instr** & **operator=** (const **Instr** &that)
- void **reset** (void)
  reset this instruction
- bool **isvalid** (void) const
  the instruction valid bit
- void **setiip** (const t_octa ip)
- t_octa **getiip** (void) const
  the instruction ip
- void **setsip** (const t_octa sip)
- t_octa **getsip** (void) const
  the speculative ip
- bool **getsfl** (void) const
  the speculative flag
- void **sethist** (const t_octa hist)
- t_octa **gethist** (void) const
  the branch history
- void **setphst** (const t_octa phst)
- t_octa **getphst** (void) const
  the predicate history
- long **getslot** (void) const
  the instruction slot
- bool **getstop** (void) const
  the instruction stop bit
- string **getgroup** (void) const
  the instruction group
- t_octa **getdata** (void) const
  the instruction data

- t_octa **getextd** (void) const
  the extension data
- t_iopc **getiopc** (void) const
  the instruction opcode
- t_unit **getbunit** (void) const
  the instruction bundle unit
- t_unit **getfunit** (void) const
  the instruction functional unit
- t_unit **getsunit** (void) const
  the instruction slot unit
- bool **getldb** (void) const
  true if the instruction is a load
- bool **getstb** (void) const
  true if the instruction is a store
- bool **isbr** (void) const
  true if the instruction is a branch
- bool **isnop** (void) const
  true if the instruction is a nop
- t_fpcomp **getfpcomp** (void) const
  the floating point status completer
- t_octa **getimmv** (const long index) const
  the instruction immediate value
- bool **ispnum** (const **Rid** &rid) const
  true if the predicate matches the rid
- **Rid getpnum** (void) const
  the predicate register number
- void **setpnum** (const **Rid** &pnum)
- bool **issnum** (const **Rid** &rid) const
  true if one source matches the rid
- **Rid getsnum** (const long index) const
- void **setsnum** (const long index, const **Rid** &snum)
- bool **isdnum** (const **Rid** &rid) const
  true if one destination matches the rid
- **Rid getdnum** (const long index) const

- void **setdnum** (const long index, const **Rid** &dnum)
- **Rpm getrrpm** (const long index) const

  the rid pair map by index

- **Operand getoper** (void) const

  the register operand

- **Result getresl** (void) const

  the result registers

- bool **ispred** (void) const

  true if the instruction is predicated

- bool **ispgen** (void) const

  true if the instruction generates predicates

- void **decode** (t_unit unit, long slot, bool bstp, t_octa inst)
- void **decode** (t_unit unit, long slot, bool bstp, t_octa inst, t_octa extd)
- string **recode** (void) const

  a string representation by doing a recoding

Static Public Member Functions

- string **tostr** (const long opcd)

  the string representation of an instruction opcode

- bool **isvig** (const string &s)

  true if the instruction group is valid

Protected Attributes

- bool **d_valid**

  valid bit

- t_octa **d_iip**

  the instruction ip

- t_octa **d_sip**

  the speculative ip

- bool **d_sfl**

  the speculative flag

- t_octa **d_hist**

  the branch history

- t_octa **d_phst**

the predicate history

- t_unit **d_bunit**
  instruction bundle unit
- t_unit **d_funit**
  instruction functional unit
- string **d_group**
  instruction group
- long **d_slot**
  the slot index
- bool **d_stop**
  stop bit
- bool **d_brch**
  the branch bit
- t_octa **d_inst**
  encoded instruction
- t_octa **d_extd**
  extension data
- t_byte **d_code**
  major group opcode
- t_iopc **d_opcd**
  detailed opcode
- bool **d_ildb**
  the load bit
- bool **d_istb**
  the store bit
- t_octa **d_immv** [IA_MSRC]
  the immediate value
- t_mhint **d_mhint**
  move to branch hint
- t_ihint **d_ihint**
  move and branch predict completer
- t_bphint **d_bphint**
  ip-relative predict hint

- t_ldhint **d_lhint**
  load hint
- t_sthint **d_shint**
  store hint
- t_phint **d_phint**
  branch prefetch hint
- t_bhint **d_bhint**
  branch direction hint
- t_chint **d_chint**
  branch cache deallocation hint
- t_lfhint **d_lfhint**
  line prefetch hint
- **Rid d_rprd**
  the predicate register
- **Rid d_rsrc** [IA_MSRC]
  the source register
- **Rpm d_rrpm** [IA_MRPM]
  the rid pair violation

2.15.1 Detailed Description

The **Instr**(p. 35) class is the instruction decode/recode for the IA64 ISA. The class holds as much as information that can be derived from the construction. The instruction holds an opcode that indicates the instruction type. The unit, slot and stop bit are also part of the instruction information. The bundle ip for that instruction is also stored. **Register**(p. 87) source and destination are store as rid (register id) object. Finally, instruction specific hints are stored as well. An instruction is generally constructed from the bundle object. The default instruction is an invalid instruction as indicated by a valid bit. Alternatively, an instruction can be built by giving the unit, slot number, stop bit flag and a 41 bit coding represented by an octa. Special long instruction uses a second field to pass the extension. Numerous methods are available to get the instruction information. A special method called 'recode' build a string a representation of that instruction as specified by the ISA. An invalid instruction can be marked by using the 'decode' method with the proper parameters. Trying to decode an invalid instruction produces an exception.

2.15.2 Constructor & Destructor Documentation

2.15.2.1 iato::Instr::Instr (t_unit *unit*, long *slot*, bool *bstp*, t_octa *inst*)

create an instruction by type/slot/stop and data

Parameters:

unit the instruction unit

slot the instruction slot

bstp the stop bit flag

inst the encoded instruction

2.15.2.2 iato::Instr::Instr (t_unit *unit*, long *slot*, bool *bstp*, t_octa *inst*, t_octa *extd*)

create an instruction by type/slot/stop data and extension

Parameters:

unit the instruction unit

slot the instruction slot

bstp the stop bit flag

inst the encoded instruction

extd the instruction extension

2.15.2.3 iato::Instr::Instr (const Instr & *that*)

copy construct an instruction

Parameters:

that the instruction to copy

2.15.3 Member Function Documentation

2.15.3.1 void iato::Instr::decode (t_unit *unit*, long *slot*, bool *bstp*, t_octa *inst*, t_octa *extd*)

set and decode an instruction with extensiion

Parameters:

unit the instruction unit

slot the instruction slot

bstp the stop bit flag

inst the encoded instruction

extd the extended data

2.15.3.2 void iato::Instr::decode (t_unit *unit*, long *slot*, bool *bstp*, t_octa *inst*)

set and decode an instruction

Parameters:

unit the instruction unit

slot the instruction slot

bstp the stop bit flag

inst the encoded instruction

2.15.3.3 Rid iato::Instr::getdnum (const long *index*) const

get the destination register number by index

Parameters:
index the destination operand index

2.15.3.4 Rid iato::Instr::getsnum (const long *index*) const

get the source register number by index

Parameters:
index the source operand index

2.15.3.5 Instr& iato::Instr::operator= (const Instr & *that*)

assign an instruction to this one

Parameters:
that the instruction to assign

2.15.3.6 void iato::Instr::setdnum (const long *index*, const Rid & *dnum*)

set the destination register number

Parameters:
index the source index
dnum the register to set

2.15.3.7 void iato::Instr::sethist (const t_octa *hist*)

set the branch history

Parameters:
hist the history to set

2.15.3.8 void iato::Instr::setiip (const t_octa *ip*)

set the instruction ip

Parameters:
ip the instruction ip

2.15.3.9 void iato::Instr::setphst (const t_octa *phst*)

set the predicate history

Parameters:
phst the history to set

2.15.3.10 void iato::Instr::setpnum (const Rid & *pnum*)

set the predicate register number

Parameters:
pnum the register to set

2.15.3.11 void iato::Instr::setsip (const t_octa *sip*)

set the speculative ip

Parameters:
sip the speculative ip

2.15.3.12 void iato::Instr::setsnum (const long *index*, const Rid & *snum*)

set the source register number

Parameters:
index the source index
snum the register to set

The documentation for this class was generated from the following file:

- Instr.hpp

2.16 iato::Interrupt Class Reference

`#include <Interrupt.hpp>`

Inherits **iato::Exception**.

Public Member Functions

- **Interrupt** (void)
  create a default interrupt
- **Interrupt** (t_icode code, const string &reason)
- **Interrupt** (t_icode code, const string &reason, const t_octa ip)
- **Interrupt** (t_icode code, const string &reason, const t_octa ip, const long slot)
- **Interrupt** (t_icode code, const **Instr** &inst)
- **Interrupt** (t_icode code, const **Instr** &inst, const string &reason)
- **Interrupt** (const **Interrupt** &that)
- **Interrupt** & **operator=** (const **Interrupt** &that)
- void **reset** (void)
  reset this interrupt
- bool **isvalid** (void) const
  true if the interrupt is valid
- bool **isabort** (void) const
  true if the interrupt is abort
- bool **isinter** (void) const
  true if the interrupt is inter
- bool **isfault** (void) const
  true if the interrupt is a fault
- bool **istrap** (void) const
  true if the interrupt is a trap
- bool **isexec** (void) const
  true if the interrupt is an execution one
- void **setexec** (const bool flag)
- t_icode **getcode** (void) const
  the interruption code
- void **setinst** (const **Instr** &inst)
- **Instr getinst** (void) const
  the interrupt instruction
- void **setip** (const t_octa ip)
- void **setip** (const t_octa ip, const long slot)
- t_octa **getip** (void) const

the offending ip

- long **getslot** (void) const

  the offending slot

- t_octa **getiim** (void) const

  the interrupt immediate value

- void **print** (void) const

  print the interrupt message

2.16.1 Detailed Description

The **Interrupt**(p. 43) class is a special class of exception that is used to model the IA64 interruption. By itself, an interrupt is not different than an exception, except that an instruction (the offending one) can be recorded. There are four classes of interrupts, namely, aborts, interrupt, faults and traps. The interruption priority is given by its code as recordered in the enumeration type. Note that the IA32 interrupt are not defined here. This class performs also the interrupt decoding by computing (on demand) the interrupt vector, the IIP, the IFA, the IIM, and the ISR. Note that some bit in the ISR are not necessarely set here since they depends on the execution context (ni bit for example). All bits that are context dependent are set to 0. In the extreme case that the interrupt comes from a place where no instruction is available, some extra field are provided to help in resolving the source or error. The base line information is the ip. In this case, the instruction record is invalid.

2.16.2 Constructor & Destructor Documentation

2.16.2.1 iato::Interrupt::Interrupt (t_icode *code*, const string & *reason*)

create a new interrupt by type and reason

Parameters:

code the interrupt code

reason the interrupt reason

2.16.2.2 iato::Interrupt::Interrupt (t_icode *code*, const string & *reason*, const t_octa *ip*)

create a new interrupt by type, reason and ip

Parameters:

code the interrupt code

reason the interrupt reason

ip the offending ip

2.16.2.3 iato::Interrupt::Interrupt (t_icode *code*, const string & *reason*, const t_octa *ip*, const long *slot*)

create a new interrupt by type, reason and ip and slot

Parameters:

code the interrupt code

reason the interrupt reason

ip the offending ip

slot the offending slot

2.16.2.4 iato::Interrupt::Interrupt (t_icode *code*, const Instr & *inst*)

create a new interrupt by type and instruction

Parameters:

code the interrupt code

inst the offending instruction

2.16.2.5 iato::Interrupt::Interrupt (t_icode *code*, const Instr & *inst*, const string & *reason*)

create a new interrupt by type, instruction and reason

Parameters:

code the interrupt code

inst the offending instruction

reason the interrupt reason

2.16.2.6 iato::Interrupt::Interrupt (const Interrupt & *that*)

copy construct this exception

Parameters:

that the interrupt to copy

2.16.3 Member Function Documentation

2.16.3.1 Interrupt& iato::Interrupt::operator= (const Interrupt & *that*)

assign an interrupt to this one

Parameters:

that the interrupt to assign

2.16.3.2 void iato::Interrupt::setexec (const bool *flag*)

set the interupt execution bit

Parameters:
flag the flag to set

2.16.3.3 void iato::Interrupt::setinst (const Instr & *inst*)

set the interrupt instruction

Parameters:
inst the instruction to set

2.16.3.4 void iato::Interrupt::setip (const t_octa *ip*, const long *slot*)

set the interrupt offending ip and slot

Parameters:
ip the offending ip
slot the offending slot

2.16.3.5 void iato::Interrupt::setip (const t_octa *ip*)

set the interrupt offending ip

Parameters:
ip the offending ip

The documentation for this class was generated from the following file:

- Interrupt.hpp

2.17 iato::Ip Class Reference

`#include <Ip.hpp>`

Public Member Functions

- **Ip** (void)
  create a default ip
- **Ip** (const t_octa value)
- **Ip** (const t_octa value, const t_octa disp)
- **Ip** (const **Ip** &that)
- **Ip** & **operator=** (const **Ip** &that)
- **Ip** & **operator++** (void)
  increase that ip by one bundle (prefix)
- **Ip operator++** (int)
  increase that ip by one bundle (postfix)
- void **reset** (void)
  reset this ip
- void **setip** (const t_octa ip)
- t_octa **getip** (void) const
  the resource ip

2.17.1 Detailed Description

The **Ip**(p. 47) is the instruction pointer resource. It is built as a resource. The sole purpose of defining the IP as a resource is to unify all special registers.

2.17.2 Constructor & Destructor Documentation

2.17.2.1 iato::Ip::Ip (const t_octa *value*)

create an ip with a value

Parameters:
value the ip value to set

2.17.2.2 iato::Ip::Ip (const t_octa *value*, const t_octa *disp*)

create an ip with a value and a displacement

Parameters:
value the ip value to set
disp the displacement to set

2.17.2.3 iato::Ip::Ip (const Ip & *that*)

copy construct this ip

Parameters:
that the ip copy

2.17.3 Member Function Documentation

2.17.3.1 Ip& iato::Ip::operator= (const Ip & *that*)

assign an ip to this one

Parameters:
that the ip to assign

2.17.3.2 void iato::Ip::setip (const t_octa *ip*)

set the ip value

Parameters:
ip the ip to set

The documentation for this class was generated from the following file:

- Ip.hpp

2.18 iato::Irt Class Reference

`#include <Irt.hpp>`

Inherits **iato::Resource**.

Public Types

- enum **t_imode**
  the irt mode

Public Member Functions

- **Irt** (void)
  create a default irt
- **Irt** (**Ctx** *ctx)
- **Irt** (**Ctx** *ctx, const string &name)
- ~**Irt** (void)
  destroy this irt
- void **reset** (void)
  reset this irt
- void **report** (void) const
  report this irt
- void **clear** (void)
  clear this irt
- void **setmode** (const t_icode code, const **t_imode** mode)
- **t_imode getmode** (const t_icode icode) const
  the irt mode by interrupt code
- void **bind** (const t_icode code, **Plugin** *plug)
- **Plugin** * **getplug** (const t_icode code) const
  the irt plugin pointer
- void **route** (const **Interrupt** &vi) const

2.18.1 Detailed Description

The **Irt**(p. 49) class is the interrupt routing table. The IRT is not part of the IA64 ISA but is defined here as a convenient way to manage interrupt in the context of emaultation or simulation. The key function of the IRT is to decide whether a particular interrupt will be serviced by the processor or by a plugin. In the case of an emulation, it is somtimes desirable to emulate some portion of the processor activity without reproducing all internal operation. For example, a system call that is the result of an intruction break, might be

serviced directly instead or recomputing the whole processor activity. This is where IRT comes into play. For each interrupt vector, a configuration code indicates what operation needs to be done. The interrupt might be ignored, rethrown (like an exception), processed (using the processor ISA) or emulated (using a plugin). When the vector is set to operate in plugin mode, control is transfered to the appropriate plugin that will handle the interrupt. When the control is set in processed mode, normal processor operation occurs as defined by the ISA.

2.18.2 Constructor & Destructor Documentation

2.18.2.1 iato::Irt::Irt (Ctx * *ctx*)

create an irt with a context

Parameters:

ctx the current context

2.18.2.2 iato::Irt::Irt (Ctx * *ctx*, const string & *name*)

create an irt with a context and name

Parameters:

ctx the current context

name the resource name

2.18.3 Member Function Documentation

2.18.3.1 void iato::Irt::bind (const t_icode *code*, Plugin * *plug*)

bind a plugin by interrupt code

Parameters:

code the interrupt code

plug the plugin to bind

2.18.3.2 void iato::Irt::route (const Interrupt & *vi*) const

route an interrupt and return a boolean status

Parameters:

vi the virtual interrupt

2.18.3.3 void iato::Irt::setmode (const t_icode *code*, const t_imode *mode*)

set the irt mode by interrupt code

Parameters:

code the interrupt code

mode the irt mode

The documentation for this class was generated from the following file:

- Irt.hpp

2.19 iato::Lru Class Reference

`#include <Lru.hpp>`

Public Member Functions

- **Lru** (const long size)
  create a lru matrix by size
- ∼**Lru** (void)
  destroy this lru matrix
- void **reset** (void)
  reset this matrix
- void **update** (const long index)
- long **getlru** (void) const
  the lru index
- long **getmru** (void) const
  the mru index

2.19.1 Detailed Description

The **Lru**(p. 52) class implements the least recently used algorithm with a n-square matrix. When an update is made, the matrix is automatically updated. The lru index is return with the getlru method. The mru index is return with the getmru method.

2.19.2 Member Function Documentation

2.19.2.1 void iato::Lru::update (const long *index*)

update an access by index

Parameters:
index the lru matrix index

The documentation for this class was generated from the following file:

- Lru.hpp

2.20 iato::MemLogic Class Reference

`#include <MemLogic.hpp>`

Inherits **iato::Resource**.

Public Member Functions

- **MemLogic** (void)

  create a default memory logic

- **MemLogic** (**Ctx** *ctx)
- ~**MemLogic** (void)

  destroy this memory logic

- void **reset** (void)

  reset this memlogic

- void **bind** (**Memory** *mem)
- void **update** (**Result** &resl)

2.20.1 Detailed Description

The **MemLogic**(p. 53) class is a class that handle the interferences between the memory and the alat. The class is updated from a result object. The ALAT is a member of the class while the memory needs to be binded.

2.20.2 Constructor & Destructor Documentation

2.20.2.1 iato::MemLogic::MemLogic (Ctx * *ctx*)

create a memory logic with a context

Parameters:

ctx the current context

2.20.3 Member Function Documentation

2.20.3.1 void iato::MemLogic::bind (Memory * *mem*)

bind a memory to the logic

Parameters:

mem the memory to bind

2.20.3.2 void iato::MemLogic::update (Result & *resl*)

update the memory, the alat and a result

Parameters:
resl the result to update

The documentation for this class was generated from the following file:

- MemLogic.hpp

2.21 iato::Memory Class Reference

`#include <Memory.hpp>`

Inherited by **iato::Segment**.

Public Member Functions

- **Memory** (void)
  create a default memory core in little endian mode
- virtual ~**Memory** (void)
  destoy this memory
- virtual void **reset** (void)=0
  reset this memory
- void **setmode** (const bool mode)
- void **setalign** (const bool mode)
- void **setprot** (const t_byte mode)
- t_byte **getprot** (void) const
  the memory protection mode
- virtual bool **isvalid** (const t_octa addr) const=0
  true if the address is valid
- virtual t_byte **readbyte** (const t_octa addr) const=0
- virtual t_byte **readexec** (const t_octa addr) const
  a byte and check for exec mode
- virtual t_word **readword** (const t_octa addr) const
- virtual t_quad **readquad** (const t_octa addr) const
- virtual t_octa **readocta** (const t_octa addr) const
- virtual **t_real readsing** (const t_octa addr) const
- virtual **t_real readdoub** (const t_octa addr) const
- virtual **t_real readxten** (const t_octa addr) const
- virtual **t_real readfill** (const t_octa addr) const
- virtual **t_real readint** (const t_octa addr) const
- virtual void **writebyte** (const t_octa addr, const t_byte byte)=0
- virtual void **writeword** (const t_octa addr, const t_word src)
- virtual void **writequad** (const t_octa addr, const t_quad src)
- virtual void **writeocta** (const t_octa addr, const t_octa src)
- virtual void **writesing** (const t_octa addr, const **t_real** &src)
- virtual void **writedoub** (const t_octa addr, const **t_real** &src)
- virtual void **writexten** (const t_octa addr, const **t_real** &src)
- virtual void **writespill** (const t_octa addr, const **t_real** &src)
- virtual void **writeint** (const t_octa addr, const **t_real** &src)
- virtual long **rdbuf** (const t_octa addr, const long blen, t_byte *buf) const
- virtual long **exbuf** (const t_octa addr, const long blen, t_byte *buf) const
- virtual long **wrbuf** (const t_octa addr, const long blen, const t_byte *buf)

Static Public Attributes

- const t_byte **PROT_NO** = 0x00

  everything protected

- const t_byte **PROT_RD** = 0x01

  read protection

- const t_byte **PROT_WR** = 0x02

  write protection

- const t_byte **PROT_EX** = 0x04

  execute protection

- const t_byte **PROT_RW** = **PROT_RD** | **PROT_WR**

  read, write protection

- const t_byte **PROT_FU** = **PROT_RD** | **PROT_WR** | **PROT_EX**

  read, write and execute protection

Protected Attributes

- bool **d_emode**

  the endian mode

- bool **d_smode**

  the swap mode

- bool **d_align**

  the align mode

- t_byte **d_prot**

  the protection mode

2.21.1 Detailed Description

The **Memory**(p. 55) class is an abstract class that defines the basic interface for any memory subsystem. Resetting the memory clear all bytes. A memory line can be copied into a buffer. The memory interface also defined the read and write endian mode. By default, the memory sub systsem operates in little endian mode. The 'setmode' method can alter this behavior. when set to true, the target memory is working in big endian mode. Note however that the memory operation with word, quad and octa are done in the host mode and the necessary conversion are made automatically. **Memory**(p. 55) protection is also defined here. By default a memory is initialized in read/write mode. Note also that a flag controls whether or not memory alignement check is done.

2.21.2 Member Function Documentation

2.21.2.1 virtual long iato::Memory::exbuf (const t_octa *addr*, const long *blen*, t_byte * *buf*) const [virtual]

read a memory line to a buffer in execute mode

Parameters:

addr the start address

blen the buffer length

buf the buffer to copy into

2.21.2.2 virtual long iato::Memory::rdbuf (const t_octa *addr*, const long *blen*, t_byte * *buf*) const [virtual]

read a memory line to a buffer

Parameters:

addr the start address

blen the buffer length

buf the buffer to copy into

2.21.2.3 virtual t_byte iato::Memory::readbyte (const t_octa *addr*) const [pure virtual]

read a byte from this memory

Parameters:

addr the address to read

Implemented in **iato::Segment** (p. 117).

2.21.2.4 virtual t_real iato::Memory::readdoub (const t_octa *addr*) const [virtual]

read an extended from this memory

Parameters:

addr the address to read

2.21.2.5 virtual t_real iato::Memory::readfill (const t_octa *addr*) const [virtual]

read a fill value from this memory

Parameters:

addr the address to read

2.21.2.6 virtual t_real iato::Memory::readint (const t_octa *addr*) const [virtual]

read a integer value from this memory

Parameters:
addr the address to read

2.21.2.7 virtual t_octa iato::Memory::readocta (const t_octa *addr*) const [virtual]

read an octa from this memory

Parameters:
addr the address to read

2.21.2.8 virtual t_quad iato::Memory::readquad (const t_octa *addr*) const [virtual]

read a quad from this memory

Parameters:
addr the address to read

2.21.2.9 virtual t_real iato::Memory::readsing (const t_octa *addr*) const [virtual]

read an extended from this memory

Parameters:
addr the address to read

2.21.2.10 virtual t_word iato::Memory::readword (const t_octa *addr*) const [virtual]

read a word from this memory

Parameters:
addr the address to read

2.21.2.11 virtual t_real iato::Memory::readxten (const t_octa *addr*) const [virtual]

read an extended from this memory

Parameters:
addr the address to read

2.21.2.12 void iato::Memory::setalign (const bool *mode*)

set the memory alignement mode

Parameters:
mode the alignment mode (true = check)

2.21.2.13 void iato::Memory::setmode (const bool *mode*)

set the memory endian mode

Parameters:
mode the memory endian mode (true = msb)

2.21.2.14 void iato::Memory::setprot (const t_byte *mode*)

set the memory protection mode

Parameters:
mode the mode to set

2.21.2.15 virtual long iato::Memory::wrbuf (const t_octa *addr*, const long *blen*, const t_byte * *buf*) [virtual]

write a memory line to a buffer

Parameters:
addr the start address
blen the buffer length
buf the buffer to copy into

2.21.2.16 virtual void iato::Memory::writebyte (const t_octa *addr*, const t_byte *byte*) [pure virtual]

write a byte at a certain address

Parameters:
addr the address to write the byte
byte the byte to write

Implemented in **iato::Segment** (p. 118).

2.21.2.17 virtual void iato::Memory::writedoub (const t_octa *addr*, const t_real & *src*) [virtual]

write a double starting at a certain address

Parameters:
addr the address to write the byte
src the real to write

2.21.2.18 virtual void iato::Memory::writeint (const t_octa *addr*, const t_real & *src*) [virtual]

write an integer val starting at a certain address

Parameters:

addr the address to write the byte

src the real to write

2.21.2.19 virtual void iato::Memory::writeocta (const t_octa *addr*, const t_octa *src*) [virtual]

write an octa starting at a certain address

Parameters:

addr the address to write the byte

src the 8 bytes to write

2.21.2.20 virtual void iato::Memory::writequad (const t_octa *addr*, const t_quad *src*) [virtual]

write a quad starting at a certain address

Parameters:

addr the address to write the byte

src the 4 bytes to write

2.21.2.21 virtual void iato::Memory::writesing (const t_octa *addr*, const t_real & *src*) [virtual]

write a single starting at a certain address

Parameters:

addr the address to write the byte

src the real to write

2.21.2.22 virtual void iato::Memory::writespill (const t_octa *addr*, const t_real & *src*) [virtual]

write a spill val starting at a certain address

Parameters:

addr the address to write the byte

src the real to write

2.21.2.23 virtual void iato::Memory::writeword (const t_octa *addr*, const t_word *src*) [virtual]

write a word starting at a certain address

Parameters:

addr the address to write the byte

src the 2 bytes to write

2.21.2.24 virtual void iato::Memory::writexten (const t_octa *addr*, const t_real & *src*) [virtual]

write an extended starting at a certain address

Parameters:

addr the address to write the byte

src the real to write

The documentation for this class was generated from the following file:

- Memory.hpp

2.22 iato::Mexecute Class Reference

`#include <Mexecute.hpp>`

Inherits **iato::Aexecute**.

Public Member Functions

- **Result exec** (const **Instr** &inst, const **Operand** &oprd) const

2.22.1 Detailed Description

The **Mexecute**(p. 62) class is the M execution unit. The class is derived from the A execution unit.

2.22.2 Member Function Documentation

2.22.2.1 Result iato::Mexecute::exec (const Instr & *inst*, const Operand & *oprd*) const `[virtual]`

execute this instruction with its operand

Parameters:

inst the instruction to execute

oprd the operand object

Reimplemented from **iato::Aexecute** (p. 5).

The documentation for this class was generated from the following file:

- Mexecute.hpp

2.23 iato::Mrt Class Reference

`#include <Mrt.hpp>`

Public Types

- enum **t_mrtt**

  the memory request type

Public Member Functions

- **Mrt** (void)

  create a default mrt

- **Mrt** (const **t_mrtt** type, const t_octa addr)
- **Mrt** (const **t_mrtt** type, const t_octa addr, const bool sbit)
- **Mrt** (const **Mrt** &that)
- **Mrt** & **operator=** (const **Mrt** &that)
- void **reset** (void)

  reset this mrt

- bool **isvalid** (void) const

  true if the mrt is valid

- bool **issbit** (void) const

  return true if the speculative bit is set

- bool **isnval** (void) const

  return true if the nat bit is set

- bool **isload** (void) const

  true if the mrt is a load

- bool **isstore** (void) const

  true if the mrt is a store

- **t_mrtt gettype** (void) const

  the mrt type

- t_octa **getaddr** (void) const

  the mrt address

- t_octa **getmask** (void) const

  the mrt mask

- bool **getmofl** (void) const

  the mrt ordering flag

- void **setnval** (const bool nval)
- void **setbval** (const t_byte bval)
- t_byte **getbval** (void) const

  the mrt byte value

- void **setwval** (const t_word wval)
- t_word **getwval** (void) const

  the mrt word value

- void **setqval** (const t_quad qval)
- t_quad **getqval** (void) const

  the mrt quad value

- void **setoval** (const t_octa oval)
- t_octa **getoval** (void) const

  the mrt octa value

- void **setlval** (const **t_real** &rval)
- **t_real getlval** (void) const

  the mrt low real value

- void **sethval** (const **t_real** &rval)
- **t_real gethval** (void) const

  the mrt high real value

- void **setlrid** (const **Rid** &rid)
- **Rid getlrid** (void) const

  the load low register

- void **sethrid** (const **Rid** &rid)
- **Rid gethrid** (void) const

  the load high register

- void **setbnd** (const t_octa addr)
- void **setld** (**t_mrtt** type, const t_octa addr, const bool sbit, const **Rid** &rid)
- void **setld** (**t_mrtt** type, const t_octa addr, const bool sbit, const **Rid** &lrid, const **Rid** &hrid)
- void **setst** (**t_mrtt** type, const t_octa addr, const t_octa oval)
- void **setst** (**t_mrtt** type, const t_octa addr, const **t_real** &rval)
- void **setst** (**t_mrtt** type, const t_octa addr, const **t_real** &lval, const **t_real** &hval)
- bool **setmv** (const **Mrt** &mrt)

2.23.1 Detailed Description

The Prq class is a simple class used to define memory port request. A request is defined by a type, an address and eventually a data. Such request is generally processed by a port.

2.23.2 Constructor & Destructor Documentation

2.23.2.1 iato::Mrt::Mrt (const t_mrtt *type*, const t_octa *addr*)

create a mrt by type and address

Parameters:

type the mrt type

addr the mrt address

2.23.2.2 iato::Mrt::Mrt (const t_mrtt *type*, const t_octa *addr*, const bool *sbit*)

create a mrt by type, address and speculative bit

Parameters:

type the mrt type

addr the mrt address

sbit the speculative bit

2.23.2.3 iato::Mrt::Mrt (const Mrt & *that*)

copy construct this mrt

Parameters:

that the mrt to copy

2.23.3 Member Function Documentation

2.23.3.1 Mrt& iato::Mrt::operator= (const Mrt & *that*)

assign mrt to this one

Parameters:

that the mrt to assign

2.23.3.2 void iato::Mrt::setbnd (const t_octa *addr*)

set a bundle mrt by address

Parameters:

addr the request address

2.23.3.3 void iato::Mrt::setbval (const t_byte *bval*)

set the mrt byte value

Parameters:

bval the value to set

2.23.3.4 void iato::Mrt::sethrid (const Rid & *rid*)

set the mrt high rid

Parameters:

rid the rid to set

2.23.3.5 void iato::Mrt::sethval (const t_real & *rval*)

set the mrt high real value

Parameters:

rval the value to set

2.23.3.6 void iato::Mrt::setld (t_mrtt *type*, const t_octa *addr*, const bool *sbit*, const Rid & *lrid*, const Rid & *hrid*)

set the load information by type, address and rid pair

Parameters:

type the load type to process

addr the load address

sbit the speculative bit

lrid the low register to load

hrid the high register to load

2.23.3.7 void iato::Mrt::setld (t_mrtt *type*, const t_octa *addr*, const bool *sbit*, const Rid & *rid*)

set the load information by type, address, speculative bit and rid

Parameters:

type the load type to process

addr the load address

sbit the speculative bit

rid the register to load

2.23.3.8 void iato::Mrt::setlrid (const Rid & *rid*)

set the mrt low rid

Parameters:

rid the rid to set

2.23.3.9 void iato::Mrt::setlval (const t_real & *rval*)

set the mrt low real value

Parameters:
rval the value to set

2.23.3.10 bool iato::Mrt::setmv (const Mrt & *mrt*)

set the mrt value from a mrt and return a ordering status

Parameters:
mrt the mrt used for setting

2.23.3.11 void iato::Mrt::setnval (const bool *nval*)

set the nat value bit

Parameters:
nval the nat value to set

2.23.3.12 void iato::Mrt::setoval (const t_octa *oval*)

set the mrt octa value

Parameters:
oval the value to set

2.23.3.13 void iato::Mrt::setqval (const t_quad *qval*)

set the mrt quad value

Parameters:
qval the value to set

2.23.3.14 void iato::Mrt::setst (t_mrtt *type*, const t_octa *addr*, const t_real & *lval*, const t_real & *hval*)

set the store information by type, address and pair value

Parameters:
type the load type to process

addr the load address

lval the low real value to store

hval the high real value to store

2.23.3.15 void iato::Mrt::setst (t_mrtt *type*, const t_octa *addr*, const t_real & *rval*)

set the store information by type, address and value

Parameters:

type the load type to process

addr the load address

rval the real value to store

2.23.3.16 void iato::Mrt::setst (t_mrtt *type*, const t_octa *addr*, const t_octa *oval*)

set the store information by type, address and value

Parameters:

type the load type to process

addr the load address

oval the octa value to store

2.23.3.17 void iato::Mrt::setwval (const t_word *wval*)

set the mrt word value

Parameters:

wval the value to set

The documentation for this class was generated from the following file:

- Mrt.hpp

2.24 iato::Operand Class Reference

`#include <Operand.hpp>`

Public Member Functions

- **Operand** (void)

  create a default operand

- **Operand** (const **Operand** &that)
- **Operand** & **operator=** (const **Operand** &that)
- void **reset** (void)

  reset this operand

- void **setrid** (const long index, const **Rid** &rid)
- **Rid getrid** (const long index) const

  the operand register by index

- bool **isvalid** (void) const

  true if the operand is valid

- bool **isvalid** (const long index) const

  the operand valid bit

- void **setuvr** (const long index, const **Uvr** &uvr)
- void **setbval** (const long index, const bool value)
- bool **getbval** (const long index) const

  the operand value by index

- void **setoval** (const long index, const t_octa value)
- t_octa **getoval** (const long index) const

  the operand value by index

- void **setrval** (const long index, const **t_real** &value)
- **t_real getrval** (const long index) const

  the operand value by index

2.24.1 Detailed Description

The **Operand**(p. 69) class is a simple class that holds an instruction source operand value. The operand is associated with a register number. This means that an operand object only holds value associated with a register. During execution, immediate value that are need by a unit are taken from the decoded instruction. The operand object is primarily used during the evaluation stage.

2.24.2 Constructor & Destructor Documentation

2.24.2.1 iato::Operand::Operand (const Operand & *that*)

copy construct an operand

Parameters:
that the operand to copy

2.24.3 Member Function Documentation

2.24.3.1 Operand& iato::Operand::operator= (const Operand & *that*)

assign an operand to this one

Parameters:
that the operand to assign

2.24.3.2 void iato::Operand::setbval (const long *index*, const bool *value*) [inline]

set the operand by index and value

Parameters:
index the operand index

value the operand value

2.24.3.3 void iato::Operand::setoval (const long *index*, const t_octa *value*) [inline]

set the operand by index and value

Parameters:
index the operand index

value the operand value

2.24.3.4 void iato::Operand::setrid (const long *index*, const Rid & *rid*) [inline]

set the operand register id

Parameters:
index the register index

rid the register id

2.24.3.5 void iato::Operand::setrval (const long *index*, const t_real & *value*) [inline]

set the operand by index and value

Parameters:

index the operand index

value the operand value

2.24.3.6 void iato::Operand::setuvr (const long *index*, const Uvr & *uvr*)

set the operand by index and uvr

Parameters:

index the operand index

uvr the uvr operand value

The documentation for this class was generated from the following file:

- Operand.hpp

2.25 iato::Pfs Class Reference

`#include <Pfs.hpp>`

Public Types

- enum **t_field**

  the pfs fields

Public Member Functions

- **Pfs** (void)

  create a default pfs

- **Pfs** (const t_octa value)
- **Pfs** (const **Pfs** &that)
- **Pfs** & **operator=** (const **Pfs** &that)
- void **reset** (void)

  reset this pfs

- void **setpfs** (const t_octa pfs)
- t_octa **getpfs** (void) const

  the resource pfs

- void **setfld** (**t_field** fld, t_octa val)
- t_octa **getfld** (**t_field** fld) const

  the pfs field

2.25.1 Detailed Description

The **Pfs**(p. 72) class is the previous frame state class object. It holds all the field that are defined by the IA64 ISA. The pfs value is stored as a single field but specific method can be used to access a particular one.

2.25.2 Constructor & Destructor Documentation

2.25.2.1 iato::Pfs::Pfs (const t_octa *value*)

create a pfs with a value

Parameters:

value the pfs value to set

2.25.2.2 iato::Pfs::Pfs (const Pfs & *that*)

copy construct this pfs

Parameters:
that the pfs to copy

2.25.3 Member Function Documentation

2.25.3.1 Pfs& iato::Pfs::operator= (const Pfs & *that*)

assign a pfs to this one

Parameters:
that the pfs to assign

2.25.3.2 void iato::Pfs::setfld (t_field *fld*, t_octa *val*)

set the pfs by field and value

Parameters:
fld the pfs field
val the pfs field value

2.25.3.3 void iato::Pfs::setpfs (const t_octa *pfs*)

set the pfs value

Parameters:
pfs the pfs to set

The documentation for this class was generated from the following file:

- Pfs.hpp

2.26 iato::Plugin Class Reference

`#include <Plugin.hpp>`

Inherits **iato::Resource**.

Public Member Functions

- **Plugin** (const string &name)
- void **reset** (void)
  reset this plugin
- virtual void **apply** (void)
  apply this plugin
- virtual void **apply** (const **Interrupt** &vi)

2.26.1 Detailed Description

The **Plugin**(p. 74) class is a pseudo-abstract class that implements the plugin interface. A plugin acts as a substitute to the processor, in order to perform certain task. A typical plugin is a system call handler. When properly installed, an emulator or a simulator can trap to the plugin and perform the required task, before returning to normal mode. The plugin can operate with any environment resources. It is the plugin constructor that is responsible to grab all necessary resources needed to operate correctly. Since the plugin is a resource, the tracer can be activated to record its operation. The plugin can operate in three mode. The first takes no parameter. The second mode operates with an instruction. The third mode operates with an interrupt.

2.26.2 Constructor & Destructor Documentation

2.26.2.1 iato::Plugin::Plugin (const string & *name*)

create a plugin by name

Parameters:
name the plugin name

2.26.3 Member Function Documentation

2.26.3.1 virtual void iato::Plugin::apply (const Interrupt & *vi*) [virtual]

apply this plugin with an interrupt

Parameters:
vi the virtul interrupt

The documentation for this class was generated from the following file:

- Plugin.hpp

2.27 iato::Psr Class Reference

`#include <Psr.hpp>`

Public Types

- enum **t_field**
  the psr fields
- enum **t_trvfd**
  the psr test reserved field

Public Member Functions

- **Psr** (void)
  create a default psr
- **Psr** (const t_octa value)
- **Psr** (const **Psr** &that)
- **Psr** & **operator=** (const **Psr** &that)
- void **reset** (void)
  reset this psr
- void **setpsr** (const t_octa psr)
- t_octa **getpsr** (void) const
  the resource psr
- void **setumr** (const **Umr** &umr)
- **Umr getumr** (void) const
  the umr value
- void **setfld** (const **t_field** fld, const bool val)
- void **setfld** (const **t_field** fld, const t_byte val)
- bool **getfld** (const **t_field** fld) const
  the psr boolean field
- t_byte **getbyte** (const **t_field** fld) const
  the psr byte field
- bool **isrvfd** (const **t_trvfd** fd, const t_octa value) const
  true if the value marks a reserved field

2.27.1 Detailed Description

The **Psr**(p. 75) class is the processor status register class object. It holds all the field that are defined by the IA64 ISA. The psr value is stored as a single field but specific method can be used to access a particular one. Note that the psr holds also the user mask register as subset.

2.27.2 Constructor & Destructor Documentation

2.27.2.1 iato::Psr::Psr (const t_octa *value*)

create a psr with a value

Parameters:
value the psr value to set

2.27.2.2 iato::Psr::Psr (const Psr & *that*)

copy construct this psr

Parameters:
that the psr to copy

2.27.3 Member Function Documentation

2.27.3.1 Psr& iato::Psr::operator= (const Psr & *that*)

assign a psr to this one

Parameters:
that the psr to assign

2.27.3.2 void iato::Psr::setfld (const t_field *fld*, const t_byte *val*)

set the psr by field and value

Parameters:
fld the psr field
val the psr field value

2.27.3.3 void iato::Psr::setfld (const t_field *fld*, const bool *val*)

set the psr by field and value

Parameters:
fld the psr field
val the psr field value

2.27.3.4 void iato::Psr::setpsr (const t_octa *psr*)

set the psr value

Parameters:
psr the psr to set

2.27.3.5 void iato::Psr::setumr (const Umr & *umr*)

set the umr value

Parameters:
umr the umr value to set

The documentation for this class was generated from the following file:

- Psr.hpp

2.28 iato::Record Class Reference

`#include <Record.hpp>`

Public Types

- enum **t_rctp**

  the record type

Public Member Functions

- **Record** (void)

  create a default record

- **Record** (const string &rsrc)
- **Record** (const string &rsrc, const **Bundle** &bndl)
- **Record** (const string &rsrc, const **Instr** &inst)
- **Record** (const string &rsrc, const **Instr** &inst, const bool flag)
- **Record** (const string &rsrc, const **Instr** &inst, const bool flag, const t_octa tg)
- **Record** (const string &rsrc, const **Result** &resl, const long index)
- **Record** (const string &rsrc, const **Operand** &oprd, const long index)
- **Record** (const string &rsrc, const t_octa ip, const long ridx)
- **Record** (const **Record** &that)
- **Record** & **operator=** (const **Record** &that)
- bool **operator**< (const **Record** &rcd) const

  true if a record in in order

- void **reset** (void)

  reset this record

- void **setname** (const string &rsrc)
- string **getname** (void) const

  the record source name

- void **settype** (**t_rctp** type)
- **t_rctp gettype** (void) const

  the record type

- bool **isignore** (void) const

  true if the record is of type ignore

- void **setrdta** (const t_octa addr, const long rlen)
- void **setbndl** (const **Bundle** &bndl)
- void **setinst** (const **Instr** &inst)
- void **setcanc** (const **Instr** &inst)
- void **setbr** (const **Instr** &inst, const bool flg, const t_octa tg)
- void **setresl** (const **Result** &resl, const long index)
- void **setoprd** (const **Operand** &oprd, const long index)

- void **setrchk** (const t_byte lreg, const t_byte pnum, const t_octa data)
- void **setrchk** (const t_byte lreg, const t_byte pnum, const t_octa data1, const t_octa data2)
- void **setrcda** (const t_octa ip, const long ridx)
- void **setrmem** (const t_octa addr, const t_octa oval, const t_byte size)
- void **setwmem** (const t_octa addr, const t_octa oval, const t_byte size)
- void **setalat** (const t_word tag, const t_octa addr, const t_byte size, const bool add)
- string **repr** (void) const

  a string representation of this record

- void **print** (void) const

  print the record information

- t_lreg **chklreg** (void) const

  the record check register type

- long **chkpnum** (void) const

  the record check register number

- bool **chkbval** (void) const

  the record check boolean value

- t_octa **chkoval** (void) const

  the record check octa value

- **t_real chkrval** (void) const

  the record check real value

- void **rcdwr** (int fd) const
- void **rcdrd** (int fd)
- **Instr getinst** (void) const

  an instruction from an instruction record

- bool **iscancel** (void) const

  true if the instruction is canceled

- bool **istip** (void) const

  true if the instruction has the tip set

- **Bundle getbnd** (void) const

  a bundle from a bundle record

- t_octa **gettip** (void) const

  the target ip

Static Public Member Functions

- **t_rctp totype** (const string &s)

2.28.1 Detailed Description

The **Record**(p. 78) class is the base class of the trace system. A record is a piece of information that needs to be recorded for future analysis. A record has a name, a source and a value.

2.28.2 Constructor & Destructor Documentation

2.28.2.1 iato::Record::Record (const string & *rsrc*)

create a default record by source

Parameters:

rsrc the record source

2.28.2.2 iato::Record::Record (const string & *rsrc*, const Bundle & *bndl*)

create a record by source and bundle

Parameters:

rsrc the record source

bndl the bundle object

2.28.2.3 iato::Record::Record (const string & *rsrc*, const Instr & *inst*)

create a record by source and instruction

Parameters:

rsrc the record source

inst the instruction object

2.28.2.4 iato::Record::Record (const string & *rsrc*, const Instr & *inst*, const bool *flag*)

create a record by source, instruction and exec flag

Parameters:

rsrc the record source

inst the instruction object

flag the execute flag (as ooposed to cancel)

2.28.2.5 iato::Record::Record (const string & *rsrc*, const Instr & *inst*, const bool *flag*, const t_octa *tg*)

create a record by source, instruction, cancel flag and target ip

Parameters:

rsrc the record source

inst the instruction object

flag the cancel flag

tg the target value

2.28.2.6 iato::Record::Record (const string & *rsrc*, const Result & *resl*, const long *index*)

create a record by source, result and index

Parameters:

rsrc the record source

resl the result object

index the result index

2.28.2.7 iato::Record::Record (const string & *rsrc*, const Operand & *oprd*, const long *index*)

create a record by source, operand and index

Parameters:

rsrc the record source

oprd the operand object

index the operand index

2.28.2.8 iato::Record::Record (const string & *rsrc*, const t_octa *ip*, const long *ridx*)

create a record by source, ip and allocated index

Parameters:

rsrc the record source

ip the record ip

ridx the record index

2.28.2.9 iato::Record::Record (const Record & *that*)

copy construct this record

Parameters:

that the record to copy

2.28.3 Member Function Documentation

2.28.3.1 Record& iato::Record::operator= (const Record & *that*)

assign a record to this one

Parameters:
that the record to assign

2.28.3.2 void iato::Record::rcdrd (int *fd*)

read and update a record from a file

Parameters:
fd the file descriptor

2.28.3.3 void iato::Record::rcdwr (int *fd*) const

write the record info a file

Parameters:
fd the file descriptor

2.28.3.4 void iato::Record::setalat (const t_word *tag*, const t_octa *addr*, const t_byte *size*, const bool *add*)

set the record with an alat action

Parameters:
tag the tag
addr the memory address
size the access size
add the add flag bit

2.28.3.5 void iato::Record::setbndl (const Bundle & *bndl*)

set the record with a bundle

Parameters:
bndl the bundle to set

2.28.3.6 void iato::Record::setbr (const Instr & *inst*, const bool *flg*, const t_octa *tg*)

set the record with a branch instruction

Parameters:
inst the instruction to set
flg the cancel flag
tg the branch target value

2.28.3.7 void iato::Record::setcanc (const Instr & *inst*)

set the record with a cancel instruction

Parameters:

inst the instruction to set

2.28.3.8 void iato::Record::setinst (const Instr & *inst*)

set the record with an instruction

Parameters:

inst the instruction to set

2.28.3.9 void iato::Record::setname (const string & *rsrc*)

set the record source name

Parameters:

rsrc the record source name

2.28.3.10 void iato::Record::setoprd (const Operand & *oprd*, const long *index*)

set the record with an operand and index

Parameters:

oprd the operand to set

index the operand index

2.28.3.11 void iato::Record::setrcda (const t_octa *ip*, const long *ridx*)

set the record with an ip and index

Parameters:

ip the record ip

ridx the record index

2.28.3.12 void iato::Record::setrchk (const t_byte *lreg*, const t_byte *pnum*, const t_octa *data1*, const t_octa *data2*)

set a record with checking data for floatting point

Parameters:

lreg the register type

pnum the register number

data1 the high part floating register value

data2 the low part floatting register value

2.28.3.13 void iato::Record::setrchk (const t_byte *lreg*, const t_byte *pnum*, const t_octa *data*)

set a record with checking data

Parameters:

lreg the register type

pnum the register number

data the register value

2.28.3.14 void iato::Record::setrdta (const t_octa *addr*, const long *rlen*)

set a record read transaction ackowldge

Parameters:

addr the request address

rlen the request length

2.28.3.15 void iato::Record::setresl (const Result & *resl*, const long *index*)

set the record with a result and index

Parameters:

resl the result to set

index the result index

2.28.3.16 void iato::Record::setrmem (const t_octa *addr*, const t_octa *oval*, const t_byte *size*)

set the record with a memory read access information

Parameters:

addr the memory address

oval the octa value

size the access size

2.28.3.17 void iato::Record::settype (t_rctp *type*)

set the record type

Parameters:

type the record type to set

2.28.3.18 void iato::Record::setwmem (const t_octa *addr*, const t_octa *oval*, const t_byte *size*)

set the record with a memory write access information

Parameters:

addr the memory address

oval the octa value

size the access size

2.28.3.19 t_rctp iato::Record::totype (const string & *s*) [static]

convert a string to a record type

Parameters:

s the type string to convert

The documentation for this class was generated from the following file:

- Record.hpp

2.29 iato::Record::t_tynm Struct Reference

a pair of record type and name

`#include <Record.hpp>`

2.29.1 Detailed Description

a pair of record type and name

The documentation for this struct was generated from the following file:

- Record.hpp

2.30 iato::Register Class Reference

`#include <Register.hpp>`

Inherits **iato::Resource**.

Public Member Functions

- **Register** (void)

  create a default register bank

- **Register** (**Ctx** *ctx)
- **Register** (**Ctx** *ctx, const string &name)
- ~**Register** (void)

  destroy this register bank

- void **reset** (void)

  reset the register

- void **report** (void) const

  report this register bank

- void **write** (t_lreg lreg, const long index, const t_octa value)
- void **write** (t_lreg lreg, const long index, const bool value)
- void **write** (t_lreg lreg, const long index, const **t_real** &value)
- void **write** (const **Result** &resl)
- t_octa **getoval** (t_lreg lreg, const long index) const
- bool **getbval** (t_lreg lreg, const long index) const
- **t_real getrval** (t_lreg lreg, const long index) const
- t_octa **getoval** (const **Rid** &rid) const
- bool **getbval** (const **Rid** &rid) const
- **t_real getrval** (const **Rid** &rid) const
- void **eval** (**Operand** &oprd) const
- bool **check** (const **Record** &rcd) const

2.30.1 Detailed Description

The **Register**(p. 87) class is a complete IA64 ISA register bank implementation. The bank holds the general purpose (and nat), the floating, the predicate, the branch, the application and machine specific registers. A generic read and write method can be used to mute these registers. This register bank is configured by default with the IA64 isa. However, for specific implementation, the size of each bank can be redefined. Note that all operation that uses an operand or a rid operates with the physical register number. Such number is assumed to be built from the logical number, the rse and any other specific implementation, like a RAT with an out-of-order machine.

2.30.2 Constructor & Destructor Documentation

2.30.2.1 iato::Register::Register (Ctx * *ctx*)

create a register bank with a context

Parameters:
ctx the current context

2.30.2.2 iato::Register::Register (Ctx * *ctx*, const string & *name*)

create a register bank with a context and name

Parameters:
ctx the current context
name the resource name

2.30.3 Member Function Documentation

2.30.3.1 bool iato::Register::check (const Record & *rcd*) const

check one record against the register file content

Parameters:
rcd the record to check

2.30.3.2 void iato::Register::eval (Operand & *oprd*) const

evaluate an operand from the bank

Parameters:
oprd the operand to evaluate

2.30.3.3 bool iato::Register::getbval (const Rid & *rid*) const

evaluate a register with a rid

Parameters:
rid the register id

2.30.3.4 bool iato::Register::getbval (t_lreg *lreg*, const long *index*) const

read a register value by type

Parameters:
lreg the register type
index the register index

2.30.3.5 t_octa iato::Register::getoval (const Rid & *rid*) const

evaluate a register with a rid

Parameters:
rid the register id

2.30.3.6 t_octa iato::Register::getoval (t_lreg *lreg*, const long *index*) const

read a register value by type

Parameters:
lreg the register type
index the register index

2.30.3.7 t_real iato::Register::getrval (const Rid & *rid*) const

evaluate a register with a rid

Parameters:
rid the register id

2.30.3.8 t_real iato::Register::getrval (t_lreg *lreg*, const long *index*) const

read a register value by type

Parameters:
lreg the register type
index the register index

2.30.3.9 void iato::Register::write (const Result & *resl*)

write a result to the register file

Parameters:
resl the result to write

2.30.3.10 void iato::Register::write (t_lreg *lreg*, const long *index*, const t_real & *value*)

write a register by type, index and value

Parameters:
lreg the register type
index the register index
value the register value

2.30.3.11 void iato::Register::write (t_lreg *lreg*, const long *index*, const bool *value*)

write a register by type, index and value

Parameters:

lreg the register type

index the register index

value the register value

2.30.3.12 void iato::Register::write (t_lreg *lreg*, const long *index*, const t_octa *value*)

write a register by type, index and value

Parameters:

lreg the register type

index the register index

value the register value

The documentation for this class was generated from the following file:

- Register.hpp

2.31 iato::Resource Class Reference

`#include <Resource.hpp>`

Inherited by **iato::Alat**, **iato::Irt**, **iato::MemLogic**, **iato::Plugin**, **iato::Register**, and **iato::Rse**.

Public Member Functions

- **Resource** (void)

  create a default resource

- **Resource** (const string &name)
- **Resource** (const **Resource** &that)
- virtual ∼**Resource** (void)

  destroy this resource

- **Resource** & **operator=** (const **Resource** &that)
- virtual void **reset** (void)=0

  reset this resource

- virtual void **report** (void) const

  report some resource information

- virtual void **setname** (const string &name)
- virtual string **getname** (void) const

  the resource name

- virtual void **setstc** (**Stat** ∗stc)
- virtual void **settrc** (**Tracer** ∗tracer)

Protected Attributes

- string **d_name**

  the resource name

- **Stat** ∗ **p_stat**

  the resource stat

- **Tracer** ∗ **p_tracer**

  the resource tracer

2.31.1 Detailed Description

The **Resource**(p. 91) class is an abstract class that is used by the environment for reference purpose. A resource is a simple unit that can be shared by others. A resource is defined by name and is generally installed in a global environment. The environment can be later queried by name to retreive a particular resource.

2.31.2 Constructor & Destructor Documentation

2.31.2.1 iato::Resource::Resource (const string & *name*)

create a new resource by name

Parameters:
name the reource name

2.31.2.2 iato::Resource::Resource (const Resource & *that*)

copy construct this resource

Parameters:
that the resource to copy

2.31.3 Member Function Documentation

2.31.3.1 Resource& iato::Resource::operator= (const Resource & *that*)

assign a resource to this one

Parameters:
that the resource to assign

2.31.3.2 virtual void iato::Resource::setname (const string & *name*) [virtual]

set the resource name

Parameters:
name the resource name to set

2.31.3.3 virtual void iato::Resource::setstc (Stat * *stc*) [virtual]

set the resource stat collector

Parameters:
stc the stat collector

2.31.3.4 virtual void iato::Resource::settrc (Tracer * *tracer*) [virtual]

set the resource tracer

Parameters:
tracer the resource tracer

The documentation for this class was generated from the following file:

- Resource.hpp

2.32 iato::Result Class Reference

`#include <Result.hpp>`

Public Types

- enum **t_rop**

  the result operation type

Public Member Functions

- **Result** (void)

  create a default result

- **Result** (const bool flag)
- **Result** (const long index, const bool value)
- **Result** (const long index, const t_octa value)
- **Result** (const long index, const **t_real** &value)
- **Result** (const **Result** &that)
- **Result** & **operator=** (const **Result** &that)
- void **reset** (void)

  reset this result

- void **setrid** (const long index, const **Rid** &rid)
- **Rid getrid** (const long index) const

  the result register by index

- void **setvalid** (const bool flag)
- bool **isvalid** (void) const

  true if the result is valid

- bool **isnone** (const long index) const

  true if the result rop is none

- bool **isreg** (const long index, t_lreg lreg) const
- bool **isreg** (t_lreg lreg) const

  true if one rid has the register type

- **t_rop getrop** (const long index) const

  the result rop by index

- t_octa **getrip** (void) const

  the result ip

- void **setrrt** (const long index, const bool flag)
- void **setaddr** (const long index, **t_rop** rop, const t_octa addr)
- t_octa **getaddr** (const long index) const

  the load/store address

- void **setimmv** (const long index, const t_octa value)
- void **setrimv** (const long index, const **t_real** value)
- t_octa **getimmv** (const long index) const

  the immediate store value by index

- **t_real getrimv** (const long index) const

  the real store value by index

- **Uvr getuvr** (const long index) const

  the result uvr value by index

- void **setnval** (const **Rid** &rid, const bool value)
- void **setbval** (const long index, const bool value)
- void **setbval** (const **Rid** &rid, const bool value)
- void **updbval** (const long index, const bool value)
- void **updbval** (const **Rid** &rid, const bool value)
- bool **getbval** (const long index) const

  the result value by index

- void **setoval** (const long index, const t_octa value)
- void **setoval** (const long index, **t_rop** rop, const t_octa value)
- void **setoval** (const **Rid** &rid, const t_octa value)
- void **updoval** (const long index, const t_octa value)
- void **updoval** (const **Rid** &rid, const t_octa value)
- t_octa **getoval** (const long index) const

  the result value by index

- void **setrval** (const long index, const **t_real** &value)
- void **setrval** (const **Rid** &rid, const **t_real** &value)
- void **updrval** (const long index, const **t_real** &value)
- void **updrval** (const **Rid** &rid, const **t_real** &value)
- **t_real getrval** (const long index) const
- void **setuval** (const **Rid** &rid, const **Uvr** &value)
- void **upduval** (const **Rid** &rid, const **Uvr** &value)
- void **setaset** (const long index, const bool value)
- void **setachk** (const long index, const bool value)
- void **setaclr** (const long index, const bool value)
- void **setspec** (const long index, const bool value)
- bool **getaset** (const long index) const

  the alat bit boolean value

- bool **getachk** (const long index) const

  the alat check bit boolean value

- bool **getaclr** (const long index) const

  the alat clear bit boolean value

- bool **getspec** (const long index) const

  the speculative bit boolean value

- void **setinv** (const long index)
- **Mrt getmrt** (void) const

  the result associated mrt

- void **update** (const **Mrt** &mrt)

2.32.1 Detailed Description

The **Result**(p. 93) class is a simple class that holds an execution unit result. For each result, a valid bit is associated to discriminate the result validity. The result object is generated by an execution unit that uses input result objects. The result class differs from the result class by the number of result and by the existence of a valid bit that indicates the execution validity, even if no result is produced.

2.32.2 Constructor & Destructor Documentation

2.32.2.1 iato::Result::Result (const bool *flag*)

create a result by valid flag

Parameters:

flag the valid flag to use

2.32.2.2 iato::Result::Result (const long *index*, const bool *value*)

create a result by index and value

Parameters:

index the result index

value the result value

2.32.2.3 iato::Result::Result (const long *index*, const t_octa *value*)

create a result by index and value

Parameters:

index the result index

value the result value

2.32.2.4 iato::Result::Result (const long *index*, const t_real & *value*)

create a result by index and value

Parameters:

index the result index

value the result value

2.32.2.5 iato::Result::Result (const Result & *that*)

copy construct an result

Parameters:
that the result to copy

2.32.3 Member Function Documentation

2.32.3.1 t_real iato::Result::getrval (const long *index*) const [inline]

Returns:
the result value by index

Parameters:
index the result index

2.32.3.2 bool iato::Result::isreg (const long *index*, t_lreg *lreg*) const

return true if a rid as the right type

Parameters:
index the register index
lreg the register type

2.32.3.3 Result& iato::Result::operator= (const Result & *that*)

assign an result to this one

Parameters:
that the result to assign

2.32.3.4 void iato::Result::setachk (const long *index*, const bool *value*) [inline]

set the alat check by index

Parameters:
index the result index
value the value to set

2.32.3.5 void iato::Result::setaclr (const long *index*, const bool *value*) [inline]

set the alat clear by index

Parameters:
index the result index
value the value to set

2.32.3.6 void iato::Result::setaddr (const long *index*, t_rop *rop*, const t_octa *addr*)

set the load/store address by index

Parameters:

index the index to set

rop the operation to set

addr the address to set

2.32.3.7 void iato::Result::setaset (const long *index*, const bool *value*) [inline]

set the alat flag by index

Parameters:

index the result index

value the value to set

2.32.3.8 void iato::Result::setbval (const Rid & *rid*, const bool *value*)

set the result by rid and value

Parameters:

rid the result rid

value the result value

2.32.3.9 void iato::Result::setbval (const long *index*, const bool *value*)

set the result by index and value

Parameters:

index the result index

value the result value

2.32.3.10 void iato::Result::setimmv (const long *index*, const t_octa *value*)

set an immediate value for store

Parameters:

index the result index

value the result value

2.32.3.11 void iato::Result::setinv (const long *index*) [inline]

set the alat index to invalidate

Parameters:

index the result rid index to invalidate

2.32.3.12 void iato::Result::setnval (const Rid & *rid*, const bool *value*)

set the result nat value by rid

Parameters:

rid the result rid to set

value the result value

2.32.3.13 void iato::Result::setoval (const Rid & *rid*, const t_octa *value*)

set the result by rid and value

Parameters:

rid the result rid

value the result value

2.32.3.14 void iato::Result::setoval (const long *index*, t_rop *rop*, const t_octa *value*)

set the result by index rop and and value

Parameters:

index the result index

rop the rop to set

value the result value

2.32.3.15 void iato::Result::setoval (const long *index*, const t_octa *value*)

set the result by index and value

Parameters:

index the result index

value the result value

2.32.3.16 void iato::Result::setrid (const long *index*, const Rid & *rid*) [inline]

set the result register id

Parameters:

index the register index

rid the register id

2.32.3.17 void iato::Result::setrimv (const long *index*, const t_real *value*)

set a real value for store

Parameters:

index the result index

value the result value

2.32.3.18 void iato::Result::setrrt (const long *index*, const bool *flag*)

set the reroute flag for a predicate register

Parameters:
index the register index
flag the reroute flag to set

2.32.3.19 void iato::Result::setrval (const Rid & *rid*, const t_real & *value*)

set the result by rid and value

Parameters:
rid the result rid
value the result value

2.32.3.20 void iato::Result::setrval (const long *index*, const t_real & *value*)

set the result by index and value

Parameters:
index the result index
value the result value

2.32.3.21 void iato::Result::setspec (const long *index*, const bool *value*) [inline]

set the speculation bit

Parameters:
index the result index
value the value to set

2.32.3.22 void iato::Result::setuval (const Rid & *rid*, const Uvr & *value*)

set the result by rid and value

Parameters:
rid the result rid
value the result value

2.32.3.23 void iato::Result::setvalid (const bool *flag*) [inline]

set the valid bit

Parameters:
flag the valid bit to set

2.32.3.24 void iato::Result::update (const Mrt & *mrt*)

update a result by mrt

Parameters:
mrt the mrt used for update

2.32.3.25 void iato::Result::updbval (const Rid & *rid*, const bool *value*)

update the result by rid and value but do not touch the rop

Parameters:
rid the result rid
value the result value

2.32.3.26 void iato::Result::updbval (const long *index*, const bool *value*)

update the result by index and value but do not touch the rop

Parameters:
index the result index
value the result value

2.32.3.27 void iato::Result::updoval (const Rid & *rid*, const t_octa *value*)

update the result by rid and value but do not touch the rop

Parameters:
rid the result rid
value the result value

2.32.3.28 void iato::Result::updoval (const long *index*, const t_octa *value*)

update the result by index and value but do not touch the rop

Parameters:
index the result index
value the result value

2.32.3.29 void iato::Result::updrval (const Rid & *rid*, const t_real & *value*)

update the result by rid and value but do not touch the rop

Parameters:
rid the result rid
value the result value

2.32.3.30 void iato::Result::updrval (const long *index*, const t_real & *value*)

update the result by index and value but do not touch the rop

Parameters:

index the result index

value the result value

2.32.3.31 void iato::Result::upduval (const Rid & *rid*, const Uvr & *value*)

update the result by rid and value but do not touch the rop

Parameters:

rid the result rid

value the result value

The documentation for this class was generated from the following file:

- Result.hpp

2.33 iato::Rid Class Reference

`#include <Rid.hpp>`

Public Member Functions

- **Rid** (void)
  create register number
- **Rid** (const **Rid** &that)
- **Rid** & **operator=** (const **Rid** &that)
- bool **operator==** (const **Rid** &rid) const
  true if two rid are equals
- void **reset** (void)
  reset this register number
- bool **isvalid** (void) const
  true if the register is valid
- bool **ispred** (void) const
  true if the rid is a valid predicate
- bool **isequal** (const **Rid** &rid) const
  true if both rid are equal physically
- void **seterdy** (const bool erdy)
- void **seterdy** (const **Rid** &rid)
- void **clrerdy** (const **Rid** &rid)
- bool **geterdy** (void) const
  the eval ready bit
- void **clrvbit** (void)
  clear the virtual bit
- void **clrvbit** (const **Rid** &rid)
- bool **getvbit** (void) const
  the virtual bit
- bool **islrnm** (void) const
  true if the rid must be logically renamed
- bool **isready** (void) const
  true if the rid is ready, invalid rid returns true
- t_lreg **gettype** (void) const
  the register type
- void **setlnum** (t_lreg type, const long lnum)

- long **getlnum** (void) const

  the logical register number

- void **setpnum** (const long pnum)
- long **getpnum** (void) const

  the physical register number

- long **getvnum** (void) const

  the register virtual number

- void **setvnum** (const long vnum)
- void **setvnum** (const long vnum, const long onum)
- long **getonum** (void) const

  the old register virtual number

- long **gettnum** (void) const

  the target register number

- void **setreg** (t_lreg type, const long lnum, const long pnum)
- string **tostring** (void) const

  a string representation of the rid

Protected Attributes

- bool **d_valid**

  the valid bit

- t_lreg **d_type**

  the register type

- long **d_lnum**

  the logical register number

- long **d_pnum**

  the physical register number

- bool **d_vbit**

  the virtual register bit

- long **d_vnum**

  the virtual register number

- long **d_onum**

  the old virtual number

- bool **d_erdy**

  the eval ready bit

2.33.1 Detailed Description

The **Rid**(p. 102) class is a simple class that binds a logical register number with a physical register number. The type of register and a valid bit is also part of this class. The rid class is primarily used used in the instruction class during decoding. The logical number is thre register number as specified by the instruction encoding. The physical number is a register number used by any renaming logic. By default the physical number equals the logical number. For example, the physical number can be used by the rse to rename some stacked or rotating register. Associated with the physical number is a virtual register bit. If set, the bit indicates that the physical register number is is a virtual register that implies a kind of indirection before matching the real physical register number. Implementation that do not perform physical register renaming, do not need this bit. The gettnum method return the virtual number if the virtual bit is set or the physical number. CAUTION: the isready method returns true if the rid is not valid or the register does not have to be renamed, otherwise it return the ready bit.

2.33.2 Constructor & Destructor Documentation

2.33.2.1 iato::Rid::Rid (const Rid & *that*)

copy construct a register number

Parameters:
that the register to copy

2.33.3 Member Function Documentation

2.33.3.1 void iato::Rid::clrerdy (const Rid & *rid*)

clear the eval ready bit if both rid are equal

Parameters:
rid the rid to compare

2.33.3.2 void iato::Rid::clrvbit (const Rid & *rid*)

clear the virtual bit by rid

Parameters:
rid the rid to compare

2.33.3.3 Rid& iato::Rid::operator= (const Rid & *that*)

assign a register number to this one

Parameters:
that the register to assign

2.33.3.4 void iato::Rid::seterdy (const Rid & *rid*)

set the eval ready bit if both rid are equal

Parameters:

rid the rid to compare

2.33.3.5 void iato::Rid::seterdy (const bool *erdy*)

set the eval ready bit

Parameters:

erdy the eval ready bit to set

2.33.3.6 void iato::Rid::setlnum (t_lreg *type*, const long *lnum*)

set the logical number by type

Parameters:

type the register type

lnum the logical register number

2.33.3.7 void iato::Rid::setpnum (const long *pnum*)

set the physical register number

Parameters:

pnum the physical register number

2.33.3.8 void iato::Rid::setreg (t_lreg *type*, const long *lnum*, const long *pnum*)

set a register by type and number

Parameters:

type the register type

lnum the logical register number

pnum the physical register number

2.33.3.9 void iato::Rid::setvnum (const long *vnum*, const long *onum*)

set the virtual register number and virtual bit

Parameters:

vnum the virtual register number

onum the old virtual number

2.33.3.10 void iato::Rid::setvnum (const long *vnum*)

set the virtual register number and virtual bit

Parameters:
vnum the virtual register number

The documentation for this class was generated from the following file:

- Rid.hpp

2.34 iato::Rpm Class Reference

`#include <Rpm.hpp>`

Public Types

- typedef **Uvr**(∗ **t_rfm**)(const **Uvr** &)
  the register mapping function

Public Member Functions

- **Rpm** (void)
  create a default rpm
- **Rpm** (const **Rid** &src, const **Rid** &dst)
- **Rpm** (const **Rpm** &that)
- **Rpm** & **operator=** (const **Rpm** &that)
- void **reset** (void)
  reset this rpm
- bool **isvalid** (void) const
  true if the rpm is valid
- void **setmap** (const **Rid** &src, const **Rid** &dst)
- void **setmap** (const **Rid** &src, const **Rid** &dst, **t_rfm** rfm)
- **Rid getsrc** (void) const
  the source rid
- **Rid getdst** (void) const
  the destination rid
- **t_rfm getrfm** (void) const
  the mapping function

2.34.1 Detailed Description

The **Rpm**(p. 107) class is a rid pair mapping class that is to map register that can eventually generates RAW or WAW legal violation. For example, the br.ret instruction can read register b0 within the same instruction group. Such case is a RAW violation but is allowed by the ISA. Therefore, the **Rpm**(p. 107) object map a branch register with the ip register.

2.34.2 Constructor & Destructor Documentation

2.34.2.1 iato::Rpm::Rpm (const Rid & *src*, const Rid & *dst*)

create a rpm with a source and destination rid

Parameters:
src the source rid

dst the destination rid

2.34.2.2 iato::Rpm::Rpm (const Rpm & *that*)

copy construct this rpm

Parameters:
that the rpm to copy

2.34.3 Member Function Documentation

2.34.3.1 Rpm& iato::Rpm::operator= (const Rpm & *that*)

assign a rpm to this one

Parameters:
that the rpm to assign

2.34.3.2 void iato::Rpm::setmap (const Rid & *src*, const Rid & *dst*, t_rfm *rfm*)

set the rpm mapping by source, destination and function mapping

Parameters:
src the source rid

dst the destination rid

rfm the rpm function mapping

2.34.3.3 void iato::Rpm::setmap (const Rid & *src*, const Rid & *dst*)

set the rpm mapping by source and destination

Parameters:
src the source rid

dst the destination rid

The documentation for this class was generated from the following file:

- Rpm.hpp

2.35 iato::Rse Class Reference

`#include <Rse.hpp>`

Inherits **iato::Resource**.

Public Member Functions

- **Rse** (void)

  create a default rse

- **Rse** (**Ctx** *ctx)
- **Rse** (**Ctx** *ctx, const string &name)
- void **reset** (void)

  reset the rse

- virtual void **flush** (void)

  flush this rse

- virtual void **setste** (const **State** &state)
- virtual **State getste** (void) const

  the current rse state

- virtual void **setsst** (const **State** &state)
- virtual **State getsst** (void) const

  the speculative rse state

- virtual bool **validate** (const **State** &state) const

  validate a state against the rse state

- virtual bool **validate** (const **Cfm** &cfm) const

  validate a cfm against the rse state

- virtual void **rename** (**Instr** &inst) const
- virtual void **preset** (const **Instr** &inst)=0
- virtual void **aftset** (const **Instr** &inst)=0
- virtual void **update** (const **Result** &resl)=0

Protected Attributes

- long **d_ngr**

  the number of logical register

- **State d_state**

  the rse state

2.35.1 Detailed Description

The **Rse**(p. 109) class is a complete **Register**(p. 87) Stack Engine class that is responsible to manage register renaming as well as spilling and filling of the stacked registers. Since the RSE is affected both by instructions and its internal state, the class separate both functions by operating virtualy with instruction or results. For exemple, with an emulator the rse is affected by instructions (like alloc) or by results (like br.ret). With a simulator, the rse must be able to operate speculatively. For this reason, the rse operates with a state object that defines the current rse state (real or speculative).

2.35.2 Constructor & Destructor Documentation

2.35.2.1 iato::Rse::Rse (Ctx * *ctx*)

create a rse with a context

Parameters:

ctx the current context

2.35.2.2 iato::Rse::Rse (Ctx * *ctx*, const string & *name*)

create a rse with a context and a name

Parameters:

ctx the current context

name the resource name

2.35.3 Member Function Documentation

2.35.3.1 virtual void iato::Rse::aftset (const Instr & *inst*) [pure virtual]

after set the rse state with an instruction

Parameters:

inst the instruction used to after set

2.35.3.2 virtual void iato::Rse::preset (const Instr & *inst*) [pure virtual]

preset the rse state with an instruction

Parameters:

inst the instruction used to preset

2.35.3.3 virtual void iato::Rse::rename (Instr & *inst*) const [virtual]

rename an instruction with the rse

Parameters:

inst the instruction to rename

2.35.3.4 virtual void iato::Rse::setsst (const State & *state*) [virtual]

set the speculative rse state by state

Parameters:
state the state to set

2.35.3.5 virtual void iato::Rse::setste (const State & *state*) [virtual]

set the rse state by state

Parameters:
state the state to set

2.35.3.6 virtual void iato::Rse::update (const Result & *resl*) [pure virtual]

update the rse state with a result

Parameters:
resl the result used to update

The documentation for this class was generated from the following file:

- Rse.hpp

2.36 iato::Rse::State Class Reference

the rse state

`#include <Rse.hpp>`

Public Member Functions

- **State** (void)
  create a default rse state
- **State** (const long ngr)
- **State** (const long ngr, const **Cfm** &cfm)
- void **reset** (void)
  reset this rse state
- bool **operator==** (const **State** &that) const
  compare a state with another one
- void **setngr** (const long ngr)
- **Cfm getcfm** (void) const
  the cfm associated with this state
- void **setcfm** (const **Cfm** &cfm)
- long **getbof** (void) const
  the state bottom of frame
- void **setbof** (const long val)
- bool **chkcfm** (const **Cfm** &cfm) const
- void **alloc** (const **Cfm** &cfm)
- void **call** (const **Cfm** &cfm)
- void **retn** (const **Cfm** &cfm)
- void **loop** (const **Cfm** &cfm)
- long **mapgr** (const long lnum, const bool rwf) const
- long **mapfr** (const long lnum) const
- long **mappr** (const long lnum) const
- **Rid maprid** (const **Rid** &rid, const bool rwf) const
- void **spill** (void)
  spill register if needed
- void **fill** (void)
  fill register if needed
- void **dump** (const string &prefix) const
  dump the rse state (for debug)

2.36.1 Detailed Description

the rse state

2.36.2 Constructor & Destructor Documentation

2.36.2.1 iato::Rse::State::State (const long *ngr*)

create a rse state with a register size

Parameters:
ngr the number of logical register

2.36.2.2 iato::Rse::State::State (const long *ngr*, const Cfm & *cfm*)

create a rse state with a register size and a cfm

Parameters:
ngr the number of logical register
cfm the cfm used to set the state

2.36.3 Member Function Documentation

2.36.3.1 void iato::Rse::State::alloc (const Cfm & *cfm*)

update the rse state for an alloc

Parameters:
cfm the cfm used to update the rse state

2.36.3.2 void iato::Rse::State::call (const Cfm & *cfm*)

update the rse state for a call

Parameters:
cfm the cfm used to update the rse state

2.36.3.3 bool iato::Rse::State::chkcfm (const Cfm & *cfm*) const

check that the cfm state equal the calling cfm

Parameters:
cfm the cfm to check

2.36.3.4 void iato::Rse::State::loop (const Cfm & *cfm*)

update the rse state for a branch loop

Parameters:
cfm the cfm used to update the rse state

2.36.3.5 long iato::Rse::State::mapfr (const long *lnum*) const

map the floating register index

Parameters:
lnum the logical register number

2.36.3.6 long iato::Rse::State::mapgr (const long *lnum*, const bool *rwf*) const

map the general register index

Parameters:
lnum the logical register number

rwf the read write flag (false = read)

2.36.3.7 long iato::Rse::State::mappr (const long *lnum*) const

map the predicate register index

Parameters:
lnum the logical register number

2.36.3.8 Rid iato::Rse::State::maprid (const Rid & *rid*, const bool *rwf*) const

map a rid by doing register renaming

Parameters:
rid the rid to rename

rwf the read-write flag

2.36.3.9 void iato::Rse::State::retn (const Cfm & *cfm*)

update the rse state for a return

Parameters:
cfm the cfm used to update the rse state

2.36.3.10 void iato::Rse::State::setbof (const long *val*)

set the state with a bottom of frame

Parameters:
val the bottom of frame value.

2.36.3.11 void iato::Rse::State::setcfm (const Cfm & *cfm*)

set the state with a cfm

Parameters:
cfm the cfm used to set the rse state

2.36.3.12 void iato::Rse::State::setngr (const long *ngr*)

set the number of general register

Parameters:
ngr the number of general registers

The documentation for this class was generated from the following file:

- Rse.hpp

2.37 iato::Segment Class Reference

`#include <Segment.hpp>`

Inherits **iato::Memory**.

Public Member Functions

- **Segment** (void)
  create an empty segment
- **Segment** (const t_long size)
- ~**Segment** (void)
  destroy this segment
- void **reset** (void)
  reset this segment
- virtual t_long **getsize** (void) const
  the segment size
- virtual void **setbase** (const t_octa addr)
- virtual t_octa **getbase** (void) const
  the base address
- virtual void **setdata** (const long blen, t_byte *buf)
- virtual void **mapdata** (const long blen, t_byte *buf)
- bool **isvalid** (const t_octa addr) const
  true if the address is valid
- t_byte **readbyte** (const t_octa addr) const
- void **writebyte** (const t_octa addr, const t_byte byte)

Protected Attributes

- t_long **d_size**
  the segment size
- t_byte * **p_data**
  the segment array
- t_octa **d_base**
  the base address

2.37.1 Detailed Description

The **Segment**(p. 116) class is an implementation of the memory core interface. The segment is defined as an array of bytes that can be accessed with a particular address. A base is used to offset the memory address.

2.37.2 Constructor & Destructor Documentation

2.37.2.1 iato::Segment::Segment (const t_long *size*)

create a new segment instance by size

Parameters:
size the segment size

2.37.3 Member Function Documentation

2.37.3.1 virtual void iato::Segment::mapdata (const long *blen*, t_byte * *buf*) [virtual]

initialize the segment data by install

Parameters:
blen the buffer length

buf the buffer to set

2.37.3.2 t_byte iato::Segment::readbyte (const t_octa *addr*) const [virtual]

read a byte from this segment

Parameters:
addr the address to read

Implements **iato::Memory** (p. 57).

2.37.3.3 virtual void iato::Segment::setbase (const t_octa *addr*) [virtual]

set the segment base address

Parameters:
addr the base segment address

2.37.3.4 virtual void iato::Segment::setdata (const long *blen*, t_byte * *buf*) [virtual]

initialize the segment data by copy

Parameters:
blen the buffer length

buf the buffer to set

2.37.3.5 void iato::Segment::writebyte (const t_octa *addr*, const t_byte *byte*) [virtual]

write a byte at a certain address

Parameters:

addr the address to write the byte

byte the byte to write

Implements **iato::Memory** (p. 59).

The documentation for this class was generated from the following file:

- Segment.hpp

2.38 iato::Stat Class Reference

`#include <Stat.hpp>`

Public Member Functions

- **Stat** (void)
  create a default stat collection
- **Stat** (const string &name)
- virtual ~**Stat** (void)
  destroy this stat object
- virtual void **reset** (void)
  reset this stat object
- virtual void **setflg** (const bool bflg, const bool iflg, const bool nflg)
- virtual bool **ismaxcc** (const t_long maxcc) const
  true if a maximum cycle count is reached
- virtual bool **ismaxic** (const t_long maxic) const
  true if a maximum instruction count is reached
- virtual void **marksc** (void)
  mark a simulation cycle
- virtual void **marksc** (const long count)
- virtual void **markes** (void)
  mark the end of the simulation
- virtual void **markpf** (const bool bflg)
- virtual void **markbp** (const bool bflg)
- virtual void **markpp** (const bool pflg)
- virtual void **markxf** (const bool xflg)
- virtual void **addbndl** (const **Bundle** &bndl)
- virtual void **addinst** (const **Instr** &inst)
- virtual void **addinst** (const **Instr** &inst, const bool cnlf)
- virtual void **addinst** (const **Instr** &inst, const bool cnlf, const bool xflg)
- virtual void **addnop** (t_unit unit)
- virtual t_long **getnnop** (void) const
  the number of nop instructions
- virtual void **printb** (void) const
  print the bundle information
- virtual void **printi** (void) const
  print the instruction information
- virtual void **printn** (void) const

print the nop information

- virtual void **summary** (void) const
  print the stat summary
- virtual void **print** (void) const
  print a stat report
- void **setos** (const string &name)
- virtual void **dump** (void) const
  dump the stat info into an output stream

Protected Attributes

- ofstream * **p_stos**
  the stat output stream
- t_long **d_stim**
  collection start time
- t_long **d_etim**
  collection end time
- t_long **d_ncyc**
  total number of cycles
- t_long **d_nbnd**
  total number of bundles
- t_long **d_nins**
  total number of instructions
- t_long **d_nuis**
  total number of usefull instructions
- t_long **d_nprd**
  total number of predicated instruction
- t_long **d_nbpd**
  number of predicated non branch instruction
- t_long **d_ncan**
  total number of cancel instruction
- t_long **d_nbcn**
  total number of cancel non branch instruction
- t_long **d_ntpf**

total number of pipeline flushes

- t_long **d_nbpf**

  number of branch pipeline flushes

- t_long **d_nopf**

  number of other pipeline flushes

- t_long **d_npbr**

  number of predicted branches

- t_long **d_npbs**

  number of successfull branch prediction

- t_long **d_nppr**

  number of predicted predicates

- t_long **d_npps**

  number of successfull predicate prediction

- t_long **d_nxmi**

  total number of extra marked instructions

- t_long **d_bndl** [**Bundle::BN_MAXTPL**]

  the bunble distribution array

- t_long **d_inst** [OPCODE_MAX]

  the instruction distribution array

- bool **d_bflg**

  the bundle flag

- bool **d_iflg**

  the instruction flag

- bool **d_nflg**

  the nop flag

2.38.1 Detailed Description

The **Stat**(p. 119) class is a collection class that can be used to collect various isa statistics and even more. The class operates by filling various information like the program name, its size and then the instruction beeing executed. The class updates its information on the fly, so at the end, all data are available.

2.38.2 Constructor & Destructor Documentation

2.38.2.1 iato::Stat::Stat (const string & *name*)

create a stat collection with a file name

Parameters:
name the file name to use

2.38.3 Member Function Documentation

2.38.3.1 virtual void iato::Stat::addbndl (const Bundle & *bndl*) [virtual]

add a new bundle for stat collection

Parameters:
bndl the bundle to add

2.38.3.2 virtual void iato::Stat::addinst (const Instr & *inst*, const bool *cnlf*, const bool *xflg*) [virtual]

add an instruction for stat collection

Parameters:
inst the instruction to add
cnlf the cancel flag
xflg the extra flag

2.38.3.3 virtual void iato::Stat::addinst (const Instr & *inst*, const bool *cnlf*) [virtual]

add an instruction for stat collection

Parameters:
inst the instruction to add
cnlf the cancel flag

2.38.3.4 virtual void iato::Stat::addinst (const Instr & *inst*) [virtual]

add an instruction for stat collection

Parameters:
inst the instruction to add

2.38.3.5 virtual void iato::Stat::addnop (t_unit *unit*) [virtual]

add a nop instruction by unit

Parameters:
unit the nop unit to add

2.38.3.6 virtual void iato::Stat::markbp (const bool *bflg*) [virtual]

mark the branch prediction stat bflg the successfull branch prediction

2.38.3.7 virtual void iato::Stat::markpf (const bool *bflg*) [virtual]

mark a pipeline flush

Parameters:
bflg the branch flag

2.38.3.8 virtual void iato::Stat::markpp (const bool *pflg*) [virtual]

mark the predicate prediction stat pflg the successfull predicate prediction

2.38.3.9 virtual void iato::Stat::marksc (const long *count*) [virtual]

mark a simulation cycle by count

Parameters:
count the number of cycle count to add

2.38.3.10 virtual void iato::Stat::markxf (const bool *xflg*) [virtual]

mark the extra stat xflg the extra stat flag

2.38.3.11 virtual void iato::Stat::setflg (const bool *bflg*, const bool *iflg*, const bool *nflg*) [virtual]

set report selection flags

Parameters:
bflg the bundle flag
iflg the instruction flag
nflg the stop flag

2.38.3.12 void iato::Stat::setos (const string & *name*)

set the dump output stream

Parameters:
name the output stream name

The documentation for this class was generated from the following file:

- Stat.hpp

2.39 iato::t_huge Class Reference

`#include <Huge.hpp>`

Public Member Functions

- **t_huge** (void)

  create a default huge number

- **t_huge** (const t_octa value)
- **t_huge** (const t_octa hip, const t_octa lop)
- **t_huge** (const **t_huge** &that)
- **t_huge** & **operator=** (const t_octa value)
- **t_huge** & **operator=** (const **t_huge** &that)
- **t_huge operator+** (const **t_huge** &x) const
- **t_huge operator** $*$ (const **t_huge** &x) const
- **t_huge operator**>> (const t_octa shc) const
- void **sethigh** (const t_octa value)
- t_octa **getlow** (void) const

  the lowest part of this huge number

- t_octa **gethigh** (void) const

  the highest part of this huge number

Static Public Attributes

- const long **HUGE_SIZE** = 16

  the huge size

2.39.1 Detailed Description

The **t_huge**(p. 124) class is the implementation of a 128 bits integer representation. Since that type is not necessarely available in almost all target machine, it has to be emulated. This class is a minimal class that is supposed to be used internally, to perform some precise computation. Tyically, it is used to support the xmpy and xma instructions. The class can be constructed with normal octa value or with two octa value to full define it. the lowest and highest representation can be obtained with the 'getlow' and 'gethigh' methods.

2.39.2 Constructor & Destructor Documentation

2.39.2.1 iato::t_huge::t_huge (const t_octa *value*)

create a huge number with an octa

Parameters:

value the value to use

2.39.2.2 iato::t_huge::t_huge (const t_octa *hip*, const t_octa *lop*)

create a huge number with an two octa

Parameters:

hip the high part to assign

lop the low part to assign

2.39.2.3 iato::t_huge::t_huge (const t_huge & *that*)

copy construct a huge with a huge

Parameters:

that the huge value to copy

2.39.3 Member Function Documentation

2.39.3.1 t_huge iato::t_huge::operator ∗ (const t_huge & *x*) const

multiply one huge with another one

Parameters:

x the huge operand

2.39.3.2 t_huge iato::t_huge::operator+ (const t_huge & *x*) const

add one huge with another one

Parameters:

x the huge operand

2.39.3.3 t_huge& iato::t_huge::operator= (const t_huge & *that*)

assign a huge to this one

Parameters:

that the huge to assign

2.39.3.4 t_huge& iato::t_huge::operator= (const t_octa *value*)

assign an octa value to this huge

Parameters:

value the value to assign

2.39.3.5 t_huge iato::t_huge::operator>> (const t_octa *shc*) const

shift right one huge

Parameters:
shc the shift count value

2.39.3.6 void iato::t_huge::sethigh (const t_octa *value*)

set high 64 bits of a huge

Parameters:
value the high part value

The documentation for this class was generated from the following file:

- Huge.hpp

2.40 iato::t_real Class Reference

`#include <Real.hpp>`

Public Types

- enum **t_sgfdp**

  the significand precision

- enum **t_expr**

  the exponent range

Public Member Functions

- **t_real** (void)

  create a default real with a quiet nan value.

- **t_real** (const long double value)
- **t_real** (const **t_real** &that)
- **t_real** & **operator=** (const long double value)
- **t_real** & **operator=** (const **t_real** &that)
- **t_real operator+** (const **t_real** &x) const
- **t_real** & **operator+=** (const **t_real** &that)
- **t_real operator-** (const **t_real** &x) const
- **t_real** & **operator-=** (const **t_real** &that)
- **t_real operator** ∗ (const **t_real** &x) const
- **t_real** & **operator** ∗= (const **t_real** &x)
- **t_real operator/** (const **t_real** &x) const
- **t_real** & **operator/=** (const **t_real** &x)
- bool **operator==** (const long double value) const
- bool **operator==** (const **t_real** value) const
- bool **operator**< (const **t_real** value) const
- bool **operator**<= (const **t_real** value) const
- bool **operator**> (const **t_real** value) const
- bool **operator**>= (const **t_real** value) const
- **operator long double** (void) const

  a machine value representation

- **t_real getabs** (void) const

  the absolute value

- string **repr** (void) const

  a raw representation of that real

- void **getbval** (t_byte ∗buf) const
- void **setnat** (void)

  set the real value to nat

- void **setpinf** (void)
  set the real value to positive infinity
- void **setninf** (void)
  set the real value to negative infinity
- void **setnanindefinite** (void)
  set the real to quiet nan indefinite
- bool **isnan** (void) const
  true if the number is nan
- bool **ispsz** (void) const
  true if the number is psz
- bool **isnat** (void) const
  true if the number is nat
- bool **ispinf** (void) const
  true if the number if positive infinity
- bool **isninf** (void) const
  true if the number if negative infinity
- bool **isinf** (void) const
  true if the number if positive or negative infinity
- bool **isfinite** (void) const
  true if the number is finite
- bool **isint** (void) const
  true if the number is an integer
- bool **isunorm** (void) const
  true if the number is unnormal
- bool **isnorm** (void) const
  true if the number is normal
- bool **unordered** (const **t_real** value) const
  true if the reals are unordered
- void **normalize** (void)
  normalize this real
- bool **getsign** (void) const
  the ia representation sign value
- t_quad **getexp** (void) const
  the ia representation exponent value

- t_octa **getsgfd** (void) const

  the ia representation significand value

- void **setsign** (const bool sign)
- void **setexp** (const t_quad exp)
- void **setsgfd** (const t_octa sgfd)
- void **setinteger** (const t_octa value)
- void **singleld** (const t_byte *buf)
- void **doubleld** (const t_byte *buf)
- void **extendedld** (const t_byte *buf)
- void **integerld** (const t_byte *buf)
- void **fill** (const t_byte *buf)
- void **singlest** (t_byte *dst) const
- void **doublest** (t_byte *dst) const
- void **extendedst** (t_byte *dst) const
- void **integerst** (t_byte *dst) const
- void **spill** (t_byte *dst) const
- **t_real rcpa** (void) const

  the ieee reciprocal

- **t_real rsqrt** (void) const

  the ieee reciprocal square root approximation

- void **convert** (const **t_sgfdp** sp, const **t_expr** er, const t_byte rc)

Static Public Attributes

- const long **TR_IASZ** = 11

  the ia representation size

- const long **TR_SISZ** = 4

  the single representation size

- const long **TR_DOSZ** = 8

  the long double representation size

- const long **TR_DESZ** = 10

  the double extended representation size

- const long **TR_SFSZ** = 16

  the spill fill representation size

Friends

- **t_real operator+** (const **t_real** &x, const long double y)
- **t_real operator-** (const **t_real** &x, const long double y)
- **t_real operator** $*$ (const **t_real** &x, const long double y)
- **t_real operator/** (const **t_real** &x, const long double y)
- ostream & **operator**$<<$ (ostream &s, const **t_real** &x)

  the string representation of the real

2.40.1 Detailed Description

The **t_real**(p. 127) class is the implementation of floating point representations. Two representations are used : One is the IA64 ISA representation, the other is the machine long double representation. Methods to cast from one representation to another are menber of the class.

2.40.2 Constructor & Destructor Documentation

2.40.2.1 iato::t_real::t_real (const long double *value*)

create a real with a machine value

Parameters:

value the value to use

2.40.2.2 iato::t_real::t_real (const t_real & *that*)

copy construct a real with a real

Parameters:

that the real to copy

2.40.3 Member Function Documentation

2.40.3.1 void iato::t_real::convert (const t_sgfdp *sp*, const t_expr *er*, const t_byte *rc*)

convert the infinite precision value into the selected model

Parameters:

sp the significand precision

er the exponent range

rc the rounding control mode

2.40.3.2 void iato::t_real::doubleld (const t_byte $*$ *buf*)

double precision memory to floating-point register data translation

Parameters:

buf the 8 bytes double precision source buffer,

2.40.3.3 void iato::t_real::doublest (t_byte * *dst*) const

floating-point register to double data memory translation

Parameters:
dst the 8 bytes memory to store in

2.40.3.4 void iato::t_real::extendedld (const t_byte * *buf*)

double extended precision memory to floating-point register data

Parameters:
buf the 10 bytes double precision source buffer,

2.40.3.5 void iato::t_real::extendedst (t_byte * *dst*) const

floating-point register to double extended data memory translation

Parameters:
dst the 10 bytes memory to store in

2.40.3.6 void iato::t_real::fill (const t_byte * *buf*)

fill floating-point register data translation.

Parameters:
buf the 16 bytes double precision source buffer,

2.40.3.7 void iato::t_real::getbval (t_byte * *buf*) const

get the ia byte representation

Parameters:
buf the destination buffer

2.40.3.8 void iato::t_real::integerld (const t_byte * *buf*)

integer memory to floating-point register data translation

Parameters:
buf the 8 bytes double precision source buffer,

2.40.3.9 void iato::t_real::integerst (t_byte * *dst*) const

floating-point register to integer data memory translation

Parameters:
dst the 8 bytes memory to store in

2.40.3.10 t_real iato::t_real::operator ∗ (const t_real & *x*) const

multiply one real with another one

Parameters:
x the real operand

2.40.3.11 t_real& iato::t_real::operator ∗= (const t_real & *x*)

multiply this real with another one

Parameters:
x the real to multiply with

2.40.3.12 t_real iato::t_real::operator+ (const t_real & *x*) const

add one real with another one

Parameters:
x the real operand

2.40.3.13 t_real& iato::t_real::operator+= (const t_real & *that*)

add this real with another one

Parameters:
that the real to add

2.40.3.14 t_real iato::t_real::operator- (const t_real & *x*) const

substract one real with another one

Parameters:
x the real operand to sub

2.40.3.15 t_real& iato::t_real::operator-= (const t_real & *that*)

substract this real with another one

Parameters:
that the real to substract

2.40.3.16 t_real iato::t_real::operator/ (const t_real & *x*) const

divide one real with another one

Parameters:
x the real operand

2.40.3.17 t_real& iato::t_real::operator/= (const t_real & *x*)

divide this real with another one

Parameters:
x the real to divide with

2.40.3.18 bool iato::t_real::operator< (const t_real *value*) const

compare a real with another one; less than operator

Parameters:
value the real to compare

2.40.3.19 bool iato::t_real::operator<= (const t_real *value*) const

compare a real with another one; less than or equal operator

Parameters:
value the real to compare

2.40.3.20 t_real& iato::t_real::operator= (const t_real & *that*)

assign a real to this one

Parameters:
that the real to assign

2.40.3.21 t_real& iato::t_real::operator= (const long double *value*)

assign a machine value to this real

Parameters:
value the value to assign

2.40.3.22 bool iato::t_real::operator== (const t_real *value*) const

compare a real with another one; equal operator

Parameters:
value the real to compare

2.40.3.23 bool iato::t_real::operator== (const long double *value*) const

compare a real with a machine value; equal operator

Parameters:
value the value to compare

2.40.3.24 bool iato::t_real::operator> (const t_real *value*) const

compare a real with another one; greater than operator

Parameters:
value the real to compare

2.40.3.25 bool iato::t_real::operator>= (const t_real *value*) const

compare a real with another one; greater than or equal operator

Parameters:
value the real to compare

2.40.3.26 void iato::t_real::setexp (const t_quad *exp*)

set the exponent field

Parameters:
exp the exponent value

2.40.3.27 void iato::t_real::setinteger (const t_octa *value*)

set the ia representation with an octa value

Parameters:
value the octa value to use

2.40.3.28 void iato::t_real::setsgfd (const t_octa *sgfd*)

set the significand field

Parameters:
sgfd the significand value

2.40.3.29 void iato::t_real::setsign (const bool *sign*)

set the sign field

Parameters:
sign the sign value

2.40.3.30 void iato::t_real::singleld (const t_byte * *buf*)

single precision memory to floating-point register data translation

Parameters:
buf the 4 bytes single precision source buffer,

2.40.3.31 void iato::t_real::singlest (t_byte * *dst*) const

floating-point register to single data memory translation

Parameters:
dst the 4 bytes memory to store in

2.40.3.32 void iato::t_real::spill (t_byte * *dst*) const

floating-point register to data memory translation

Parameters:
dst the 16 bytes memory to store in

2.40.4 Friends And Related Function Documentation

2.40.4.1 t_real operator * (const t_real & *x*, const long double *y*) [friend]

multiply one real with a machine one

Parameters:
x the real operand
y the machine operand

2.40.4.2 t_real operator+ (const t_real & *x*, const long double *y*) [friend]

add one real with a machine one

Parameters:
x the real operand
y the machine operand

2.40.4.3 t_real operator- (const t_real & *x*, const long double *y*) [friend]

substract one real with a machine one

Parameters:
x the real operand
y the machine operand

2.40.4.4 t_real operator/ (const t_real & *x*, const long double *y*) [friend]

divide one real with a machine one

Parameters:
x the real operand
y the machine operand

The documentation for this class was generated from the following file:

- Real.hpp

2.41 iato::Tracer Class Reference

`#include <Tracer.hpp>`

Public Types

- typedef vector< **Record** > **t_vrcd**
  record vector type
- typedef vector< **t_vrcd** * > **t_trcd**
  trace vector type

Public Member Functions

- **Tracer** (void)
  create a default tracer
- **Tracer** (**Ctx** *ctx)
- **Tracer** (const bool vflg)
- virtual ~**Tracer** (void)
  destroy this tracer
- virtual void **reset** (void)
  reset this tracer
- virtual void **setname** (const string &name)
- virtual void **addtype** (**Record::t_rctp** val)
- virtual void **newtrace** (void)
  start a new trace
- virtual void **newtraces** (const long count)
  add several empty traces
- virtual void **add** (const **Record** &rcd)
- virtual void **add** (**t_vrcd** *vrcd)
- virtual void **print** (void) const
  print the current record vector

2.41.1 Detailed Description

The **Tracer**(p. 136) class is a cycle based trace recorder. For a given cycle the tracer accumulates records until a new trace is started. Traces are accumulated in a vector and can be saved (in a binary form) in a file. The 'setname' method can be used to set that file name and open it. When used, the file is opened and a header is written. When number of traces reaches a certain threshold, the traces are saved in the file. Note that record are saved according to record type and name. Note also that traces are saved according to their cycle number.

2.41.2 Constructor & Destructor Documentation

2.41.2.1 iato::Tracer::Tracer (Ctx * *ctx*)

create a tracer with a context

Parameters:
ctx the current context

2.41.2.2 iato::Tracer::Tracer (const bool *vflg*)

create a tracer with a verbose flag

Parameters:
vflg the verbose flag

2.41.3 Member Function Documentation

2.41.3.1 virtual void iato::Tracer::add (t_vrcd * *vrcd*) [virtual]

add a record vector to the tracer

Parameters:
vrcd the record vector to add

2.41.3.2 virtual void iato::Tracer::add (const Record & *rcd*) [virtual]

add a record to the current cycle

Parameters:
rcd the record to add

2.41.3.3 virtual void iato::Tracer::addtype (Record::t_rctp *val*) [virtual]

add a record type to trace type vector

Parameters:
val the record type to add

2.41.3.4 virtual void iato::Tracer::setname (const string & *name*) [virtual]

set the tracer file name and open it

Parameters:
name the file name to set and open

The documentation for this class was generated from the following file:

- Tracer.hpp

2.42 iato::Tracer::Reader Class Reference

the trace reader class

`#include <Tracer.hpp>`

Public Member Functions

- **Reader** (const string &name)
- ∼**Reader** (void)
  destroy this trace reader
- **t_vrcd** ∗ **trcrd** (void)
  a new tracer record
- **Record** ∗ **getrcd** (void)
  a new record
- void **hdinfo** (void)
  print header information
- t_long **getbicc** (void) const
  the begin trace index
- t_long **geteicc** (void) const
  the end trace index

2.42.1 Detailed Description

the trace reader class

2.42.2 Constructor & Destructor Documentation

2.42.2.1 iato::Tracer::Reader::Reader (const string & *name*)

create a new trace reader by name

Parameters:
name the trace file name

The documentation for this class was generated from the following file:

- Tracer.hpp

2.43 iato::Umr Class Reference

`#include <Umr.hpp>`

Public Types

- enum **t_field**
  the umr fields

Public Member Functions

- **Umr** (void)
  create a default umr
- **Umr** (const t_byte value)
- **Umr** (const **Umr** &that)
- **Umr** & **operator=** (const **Umr** &that)
- void **reset** (void)
  reset this umr
- void **setumr** (const t_byte umr)
- t_byte **getumr** (void) const
  the resource umr
- void **setfld** (**t_field** fld, const bool val)
- bool **getfld** (**t_field** fld) const
  the umr field

2.43.1 Detailed Description

The **Umr**(p. 139) class is the user mask register. The UMR is a subset of the processor status register (PSR). The UMR can be modified here and later inserted into the PSR

2.43.2 Constructor & Destructor Documentation

2.43.2.1 iato::Umr::Umr (const t_byte *value*)

create a umr with a value

Parameters:
value the umr value to set

2.43.2.2 iato::Umr::Umr (const Umr & *that*)

copy construct this umr

Parameters:
that the umr to copy

2.43.3 Member Function Documentation

2.43.3.1 Umr& iato::Umr::operator= (const Umr & *that*)

assign a umr to this one

Parameters:

that the umr to assign

2.43.3.2 void iato::Umr::setfld (t_field *fld*, const bool *val*)

set the umr by field and value

Parameters:

fld the umr field

val the umr field value

2.43.3.3 void iato::Umr::setumr (const t_byte *umr*)

set the umr value

Parameters:

umr the umr to set

The documentation for this class was generated from the following file:

- Umr.hpp

2.44 iato::Uvr Class Reference

`#include <Uvr.hpp>`

Public Types

- enum **t_uvrt**

  the uvr types

Public Member Functions

- **Uvr** (void)

  create a default uvr

- **Uvr** (const **Uvr** &that)
- **Uvr** & **operator=** (const **Uvr** &that)
- void **reset** (void)

  reset this uvr

- bool **isvalid** (void) const

  true if the uvr is valid

- **t_uvrt gettype** (void) const

  the uvr type

- void **setbval** (const bool bval)
- bool **getbval** (void) const

  the boolean value

- void **setoval** (const t_octa oval)
- t_octa **getoval** (void) const

  the octa value

- void **setdval** (const t_octa oval, const bool bval)
- void **setrval** (const **t_real** rval)
- **t_real getrval** (void) const

  the real value

2.44.1 Detailed Description

The **Uvr**(p. 141) class is a simple class that defines a universal value representation. The default representation is NAV (not a value. The value can be either an octa, a combination of octa and nat bit, a boolean value or a real value.

2.44.2 Constructor & Destructor Documentation

2.44.2.1 iato::Uvr::Uvr (const Uvr & *that*)

copy construct this uvr

Parameters:
that the uvr to copy

2.44.3 Member Function Documentation

2.44.3.1 Uvr& iato::Uvr::operator= (const Uvr & *that*)

assign a uvr to this one

Parameters:
that the uvr to assign

2.44.3.2 void iato::Uvr::setbval (const bool *bval*)

set a boolean value

Parameters:
bval the boolean value to set

2.44.3.3 void iato::Uvr::setdval (const t_octa *oval*, const bool *bval*)

set a dual value (octa and nat bit)

Parameters:
oval the octa value to set
bval the nat bit to set

2.44.3.4 void iato::Uvr::setoval (const t_octa *oval*)

set an octa value

Parameters:
oval the octa value to set

2.44.3.5 void iato::Uvr::setrval (const t_real *rval*)

set a real value

Parameters:
rval the real value to set

The documentation for this class was generated from the following file:

- Uvr.hpp

Part II

ELF Library

Chapter 3

ELF Library Compound Index

3.1 IATO ELF LIBRARY Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

ELF Library Reference

4.1 iato::ElfArgs Class Reference

`#include <ElfArgs.hpp>`

Inherits **iato::ElfTable**.

Public Member Functions

- **ElfArgs** (const string &name)
- **ElfArgs** (const long argc, const char **argv)
- **ElfArgs** (const string &name, const vector< string > &argv)
- string **getname** (void) const

  the program name

4.1.1 Detailed Description

The **ElfArgs**(p. 147) class is a special class designed to build the memory image of the program arguments. For a given program, a character block is built with the total size of the arguments. Note that we are using C strings, that means the trailing null character is also included. Then the block is padded to align it with a 8 bytes boundary. Note that we store also the number of arguments and we use both data to initialize the program stack. This kind of operation is heavily ABI sensitive. You have been warned ...

4.1.2 Constructor & Destructor Documentation

4.1.2.1 iato::ElfArgs::ElfArgs (const string & *name*)

create an argument block by name

Parameters:

name the program name

4.1.2.2 iato::ElfArgs::ElfArgs (const long *argc*, const char ** *argv*)

create an argument block by arguments

Parameters:

argc the number of arguments

argv the argument vector

4.1.2.3 iato::ElfArgs::ElfArgs (const string & *name*, const vector< string > & *argv*)

create an argument block by name and arguments

Parameters:

name the program name

argv the program arguments

The documentation for this class was generated from the following file:

- ElfArgs.hpp

4.2 iato::ElfBrk Class Reference

`#include <ElfBrk.hpp>`

Inherits **iato::ElfLoad**.

Public Member Functions

- **ElfBrk** (void)

  create a new breakable memory

- **ElfBrk** (**ElfKernel** *ekp)
- void **addseg** (**ElfSegment** *seg)
- void **setbrkta** (const t_octa addr)
- t_octa **getbrkta** (void) const

  the break limit address

4.2.1 Detailed Description

The **ElfBrk**(p. 149) class is a specialized elf load object that maintain a break limit for further memory allocation. Each time a new segment is added the break limit is changed, as to reflect the change in memory access. Increasing the break limit is equivalent to increase the segment size of the containing segment. Decreasing the break limit can result in segment removal.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 iato::ElfBrk::ElfBrk (ElfKernel * *ekp*)

create a new breakable memory by parameters

Parameters:
ekp the elf kernel parameters

4.2.3 Member Function Documentation

4.2.3.1 void iato::ElfBrk::addseg (ElfSegment * *seg*) [virtual]

add a segment to this memory

Parameters:
seg the segment to add

Reimplemented from **iato::ElfLoad** (p. 163).

4.2.3.2 void iato::ElfBrk::setbrkta (const t_octa *addr*)

change the break limit address

Parameters:
addr the break limit to change

The documentation for this class was generated from the following file:

- ElfBrk.hpp

4.3 iato::ElfBsa Class Reference

`#include <ElfBsa.hpp>`

Public Member Functions

- **ElfBsa** (void)

  create a new default bsa

- **ElfBsa** (**ElfKernel** *ekp)

4.3.1 Detailed Description

The **ElfBsa**(p. 151) class is a simple segment that represents the backing store area. The class is nothing more than a constructor. The good news is that it can be constructed from the **ElfKernel**(p. 161) object.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 iato::ElfBsa::ElfBsa (ElfKernel * *ekp*)

create a new bsa by context

Parameters:

ekp the elf kernel parameters

The documentation for this class was generated from the following file:

- ElfBsa.hpp

4.4 iato::ElfChecker Class Reference

`#include <ElfChecker.hpp>`

Inherits **iato::ElfSection**.

Public Member Functions

- Checker * **getchecker** (void) const

  the checker object

4.4.1 Detailed Description

The **ElfChecker**(p. 152) class is a special section that is designed to handle the special checker section. Once the section has been read, it is possible to extract a checker object. The checker object is built by scaning the section and building register check records. Once the checker object has been built, that section can be destroyed.

The documentation for this class was generated from the following file:

- ElfChecker.hpp

4.5 iato::ElfEnvp Class Reference

`#include <ElfEnvp.hpp>`

Inherits **iato::ElfTable**.

Public Member Functions

- **ElfEnvp** (void)
  create a default elf environment
- **ElfEnvp** (const long argc, const char **argv)

4.5.1 Detailed Description

The **ElfEnvp**(p. 153) class is a special class designed to build the memory image of the program environment. For a given program, a character block is built with the total size of the environment. This class is similar to the **ElfArgs**(p. 147) that manages the program arguments. As part of the ABI, the environment array is placed above the arguement array. That means that the first pointer is visible at argv[argc+1]. Once again, this class is ABI sensitive. You have been warned.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 iato::ElfEnvp::ElfEnvp (const long *argc*, const char ** *argv*)

create an argument block

Parameters:

argc the number of arguments

argv the argument vector

The documentation for this class was generated from the following file:

- ElfEnvp.hpp

4.6 iato::ElfExec Class Reference

`#include <ElfExec.hpp>`

Inherits **iato::ElfMemory**.

Public Member Functions

- **ElfExec** (**ElfKernel** *ekp)
- void **setbrkm** (**ElfBrk** *brk)
- **ElfBrk** * **getbrkm** (void) const
  the elf breakable memory
- **ElfStack** * **getstkm** (void) const
  the elf stack memory
- **ElfBsa** * **getbsam** (void) const
  the elf bsa memory
- **ElfMap** * **getmapm** (void) const
  the elf mappable memory

4.6.1 Detailed Description

The **ElfExec**(p. 154) class is a memory object that represent the various portions of a memory process. It is built with a breakable memory, a backing store area, a stack and a mappable area. To be effectively constructed, the class needs the **ElfKernel**(p. 161) object in order to grab the stack and backing store parameters. The breakable memory can be added at any time.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 iato::ElfExec::ElfExec (ElfKernel * *ekp*)

create an elf executable memory

Parameters:
ekp the elf kernel parameters

4.6.3 Member Function Documentation

4.6.3.1 void iato::ElfExec::setbrkm (ElfBrk * *brk*)

set the breakable memory

Parameters:
brk the breakable memory to set

The documentation for this class was generated from the following file:

- ElfExec.hpp

4.7 iato::ElfImage Class Reference

`#include <ElfImage.hpp>`

Public Member Functions

- **ElfImage** (const string &name)
- **ElfImage** (**Etx** *etx, const string &name)
- **ElfImage** (**Etx** *etx, const string &name, const vector< string > &argv)
- **ElfImage** (**Etx** *etx, const long argc, const char **argv)
- ~**ElfImage** (void)

  destroy this elf image

- void **reset** (void)

  reset this elf image

- string **getname** (void) const

  the elf image name

- string **repr** (void) const

  a string representation

- bool **isvalid** (void) const

  true if the elf image is valid

- bool **isexec** (void) const

  true if the elf image is an executable

- bool **isstatic** (void) const

  true if the elf image is static

- bool **iscls64** (void) const

  true if the image is of class 64

- bool **isia64** (void) const

  true if the image is an ia64 image

- bool **ismsb** (void) const

  true if the image has msb encoding

- long **getphnum** (void) const

  the number of program headers

- long **getshnum** (void) const

  the number of section headers

- t_octa **getentry** (void) const

  the elf entry point

- string **getinterpname** (void) const

  the elf interpreter name

- **ElfInterp** ∗ **getinterp** (void) const

  the associated elf interpreter

- **ElfSection** ∗ **getscn** (const string &name) const

  a section by name

- **ElfText** ∗ **gettxt** (void) const

  all elf text sections

- **ElfLoad** ∗ **getload** (void) const

  all elf load segments

- **ElfBrk** ∗ **getbrkm** (void) const

  the elf breakable memory

- **ElfExec** ∗ **getexec** (void) const

  the elf executable image

- **ElfExec** ∗ **getimage** (void) const

  an elf executable image or throw an exception

- Checker ∗ **getchecker** (void) const

  a checker object if it exists

4.7.1 Detailed Description

The Elf class is a complete representation of an elf object. At initialization, the elf file is opened and the header is read. A determination of weather the file is a relocatable file (.o) or an executable file. For a relocatable file, the linking view of the elf file is used (aka using section). For an executable file the executable view is used (aka segments). Various methods are provided to get an idea of the file nataure. The 'isvalid' method is particulary useful to determined the object state. The 'isexec' method tells if the elf image is an executable object. The 'getimage' method is the prefered way to get an execuatble image since it prepares the complete memory representation.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 iato::ElfImage::ElfImage (const string & *name*)

create a new elf image by name

Parameters:

name the file name

4.7.2.2 iato::ElfImage::ElfImage (Etx * *etx*, const string & *name*)

create a new elf image by context and name

Parameters:

etx the elf context

name the file name

4.7.2.3 iato::ElfImage::ElfImage (Etx * *etx*, const string & *name*, const vector< string > & *argv*)

create a new elf image by context, name and arguments

Parameters:

etx the elf context

name the program name

argv the program arguments

4.7.2.4 iato::ElfImage::ElfImage (Etx * *etx*, const long *argc*, const char ** *argv*)

create a new elf image by context and arguments

Parameters:

etx the elf context

argc the number of arguments

argv the argument vector

The documentation for this class was generated from the following file:

- ElfImage.hpp

4.8 iato::ElfInterp Class Reference

`#include <ElfInterp.hpp>`

Public Member Functions

- **ElfInterp** (void)
  create a default interpreter
- **ElfInterp** (const string &name)
- virtual ~**ElfInterp** (void)
  destroy this interpreter
- virtual void **bind** (**ElfKernel** *ekp)
- virtual void **bind** (**ElfArgs** *args, **ElfEnvp** *envp, **ElfStack** *stk) const
- void **setph** (const t_octa phdr, const t_octa phent, const t_octa phnum)

Static Public Member Functions

- bool **isvalid** (const string &interp)
  true if the interpreter is supported

4.8.1 Detailed Description

The **ElfInterp**(p. 159) class is the elf interpreter. As definied by the ABI, the interpreter is responsible to setup the memory image so that a program can be executed. The memory image setup include, among other things, the preparation of the stack (with arguments), the got and plt setup, relocation and many other stuff. This implementation is the default one. It makes some assumption about the process image and mimic a standard IA64 ld.so implementation. Although this might vary from one implementation to another, it is unlikelly to change unless the libc is changed dramatically. Note that in real life, the elf interpreter is part of the libc. This means that program compiled with a special library might not work. Again, this is unlikelly to happen.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 iato::ElfInterp::ElfInterp (const string & *name*)

create an interpreter by name

Parameters:
name the emulated name

4.8.3 Member Function Documentation

4.8.3.1 virtual void iato::ElfInterp::bind (ElfArgs * *args*, ElfEnvp * *envp*, ElfStack * *stk*) const `[virtual]`

bind the arguments in a memory

Parameters:

args the program argument

envp the program environment

stk the elf stack

4.8.3.2 virtual void iato::ElfInterp::bind (ElfKernel * *ekp*) [virtual]

bind the kernel parameters

Parameters:

ekp the elf kernel parameters

4.8.3.3 void iato::ElfInterp::setph (const t_octa *phdr*, const t_octa *phent*, const t_octa *phnum*)

set the program header info at once

Parameters:

phdr the program header address

phent the program header size

phnum the number of program header

The documentation for this class was generated from the following file:

- ElfInterp.hpp

4.9 iato::ElfKernel Class Reference

`#include <ElfKernel.hpp>`

Public Member Functions

- **ElfKernel** (void)
  create a default elf kernel parameters
- **ElfKernel** (**Etx** ∗etx)
- void **setmode** (const bool mode)
- bool **getmode** (void) const
  the endian mode
- t_octa **getmapb** (void) const
  the mappable base address
- t_octa **getpgsz** (void) const
  the page size
- t_octa **getstkva** (void) const
  the top stack address
- long **getstksz** (void) const
  the stack size
- t_octa **getbsava** (void) const
  the bsa base address
- long **getbsasz** (void) const
  the bsa size
- long **getclktk** (void) const
  the times clock ticks

4.9.1 Detailed Description

The **ElfKernel**(p. 161) class is a parameters class that is used to represent the process image according to some kernel parameters. Such parameters include the page size, process stack definition as well as backing store area information. The class is used to construct other object like the stack or the bsa.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 iato::ElfKernel::ElfKernel (Etx ∗ *etx*)

create a kernel parameters by context

Parameters:
etx the elf context

4.9.3 Member Function Documentation

4.9.3.1 void iato::ElfKernel::setmode (const bool *mode*)

set the endian mode

Parameters:
mode the mode to set

The documentation for this class was generated from the following file:

- ElfKernel.hpp

4.10 iato::ElfLoad Class Reference

`#include <ElfLoad.hpp>`

Inherits **iato::ElfMemory**.

Inherited by **iato::ElfBrk**, and **iato::ElfMap**.

Public Member Functions

- **ElfLoad** (void)
  create a default load memory
- void **add** (Memory *mem)
- virtual void **addseg** (**ElfSegment** *seg)
- **ElfSegment** * **getseg** (const long index) const
  a segment by index

4.10.1 Detailed Description

The **ElfLoad**(p. 163) class is a specialized elf memory object that is built from an elf file. The class acts as an array of elf segments built with load flag. This class is similar to the elf text section array. Note that globally, the load array has all protections enabled.

4.10.2 Member Function Documentation

4.10.2.1 void iato::ElfLoad::add (Memory * *mem*) [virtual]

add a memory only if it is an elf segment

Parameters:
mem the memory to add

Reimplemented from **iato::ElfMemory** (p. 167).

4.10.2.2 virtual void iato::ElfLoad::addseg (ElfSegment * *seg*) [virtual]

add a segment to this memory

Parameters:
seg the segment to add

Reimplemented in **iato::ElfBrk** (p. 149).

The documentation for this class was generated from the following file:

- ElfLoad.hpp

4.11 iato::ElfMap Class Reference

`#include <ElfMap.hpp>`

Inherits **iato::ElfLoad**.

Public Member Functions

- **ElfMap** (void)

  create a new mappable memory

- **ElfMap** (**ElfKernel** ∗ekp)
- t_octa **findtop** (void) const

  the top available address

- t_octa **mmap** (const t_long size, const t_byte prot, **ElfSegment::t_stype** type)
- bool **munmap** (const t_octa addr, const t_long size)

4.11.1 Detailed Description

The **ElfMap**(p. 164) class is a specialized elf load object that maintain the memory mappable interface. The full 'mmap' system call is implemented via this class. The starting address for a memory mappable is defined as a parameters. The memory can be mapped or unmapped. For a memory mapping, the meory size needs to be alligned to a page size or an exception is raised.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 iato::ElfMap::ElfMap (ElfKernel ∗ *ekp*)

create a new mappable memory by parameters

Parameters:

ekp the elf kernel parameters

4.11.3 Member Function Documentation

4.11.3.1 t_octa iato::ElfMap::mmap (const t_long *size*, const t_byte *prot*, ElfSegment::t_stype *type*)

map some memory anonymously with some protection

Parameters:

size the memory size to map

prot the memory protection to use

type the memory type to map

4.11.3.2 bool iato::ElfMap::munmap (const t_octa *addr*, const t_long *size*)

unmap some memory at a certan address and size

Parameters:

addr the base memory address

size the memory size to unmap

The documentation for this class was generated from the following file:

- ElfMap.hpp

4.12 iato::ElfMemory Class Reference

`#include <ElfMemory.hpp>`

Inherited by **iato::ElfExec**, **iato::ElfLoad**, and **iato::ElfText**.

Public Member Functions

- **ElfMemory** (void)
  create a default elf memory
- ∼**ElfMemory** (void)
  destroy this elf memory
- void **reset** (void)
  reset this elf memory
- bool **isvalid** (const t_octa addr) const
  true if the address is valid
- t_byte **readbyte** (const t_octa addr) const
- t_byte **readexec** (const t_octa addr) const
- void **writebyte** (const t_octa addr, const t_byte byte)
- virtual long **length** (void) const
  the number of memories
- virtual long **find** (const t_octa addr) const
  a memory index by address
- virtual void **add** (Memory ∗mem)
- virtual Memory ∗ **getmem** (const long index) const
  a memory by index
- virtual bool **remove** (const long index)
  remove a memory by index

Protected Attributes

- vector< Memory ∗ > **d_vmem**
  the vector of memory

4.12.1 Detailed Description

The **ElfMemory**(p. 166) is a container for other elements. The class implements the memory interface and is responsible to select the rigth memory block for any operation, like read or write. The class works also with imbricated memory block, that is an elf memory can also contain other memory block.

4.12.2 Member Function Documentation

4.12.2.1 virtual void iato::ElfMemory::add (Memory * *mem*) [virtual]

add a memory to this memory array

Parameters:
mem the memory to add

Reimplemented in **iato::ElfLoad** (p. 163), and **iato::ElfText** (p. 176).

4.12.2.2 t_byte iato::ElfMemory::readbyte (const t_octa *addr*) const

read a byte from this memory

Parameters:
addr the address to read

4.12.2.3 t_byte iato::ElfMemory::readexec (const t_octa *addr*) const

read a byte from this memory and check for execute

Parameters:
addr the address to read

4.12.2.4 void iato::ElfMemory::writebyte (const t_octa *addr*, const t_byte *byte*)

write a byte at a certain address

Parameters:
addr the address to write the byte
byte the byte to write

The documentation for this class was generated from the following file:

- ElfMemory.hpp

4.13 iato::ElfSection Class Reference

`#include <ElfSection.hpp>`

Inherited by **iato::ElfChecker**.

Public Member Functions

- **ElfSection** (const string &name, void *scn, const bool mode)
- string **getname** (void) const

  the section name

- **const_iterator begin** (void) const

  a bundle iterator pointing at the beginning of the section

- **const_iterator end** (void) const

  a bundle iterator pointing at the end of the section

4.13.1 Detailed Description

The **ElfSection**(p. 168) class is a simple class that holds the data associated with a particular elf section. The class cannot be constructed directly. The elf section is described with a name, a size and a set of data.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 iato::ElfSection::ElfSection (const string & *name*, void * *scn*, const bool *mode*)

create a new elf section by name and elf section

Parameters:

name the section name

scn the elf section descriptor

mode the endian mode

The documentation for this class was generated from the following file:

- ElfSection.hpp

4.14 iato::ElfSection::const_iterator Class Reference

`#include <ElfSection.hpp>`

Public Member Functions

- **const_iterator** (void)

  create a default iterator

- **const_iterator** (const **ElfSection** *scn, const t_octa addr)
- **const_iterator** (const **const_iterator** &that)
- **const_iterator** & **operator=** (const **const_iterator** &it)
- **const_iterator** & **operator++** (void)

  move one step the iterator (prefix)

- **const_iterator operator++** (int)

  move one step the iterator (postfix)

- bool **operator==** (const **const_iterator** &it) const

  true if two iterators are equal

- bool **operator**< (const **const_iterator** &it) const

  true if two iterators are less

- bool **operator**<= (const **const_iterator** &it) const

  true if two iterators are less equal

- bool **operator**> (const **const_iterator** &it) const

  true if two iterators are greater

- bool **operator**>= (const **const_iterator** &it) const

  true if two iterators are greater equal

- Bundle **operator** * (void) const

  get a bundle from this iterator

- t_octa **getip** (void) const

  the current iterator ip

4.14.1 Detailed Description

the **const_iterator**(p. 169) class is a constant iterator for the section class. The iterator returns a bundle object per iteration. If the iterator is not valid an invalid bundle is returned.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 iato::ElfSection::const_iterator::const_iterator (const ElfSection * *scn*, const t_octa *addr*)

create an iterator with a section and address

Parameters:

scn the section to iterate

addr the address to start

4.14.2.2 iato::ElfSection::const_iterator::const_iterator (const const_iterator & *that*)

copy construct this iterator

Parameters:

that the iterator to copy

4.14.3 Member Function Documentation

4.14.3.1 const_iterator& iato::ElfSection::const_iterator::operator= (const const_iterator & *it*)

assign an iterator to this one

Parameters:

it the iterator to assign

The documentation for this class was generated from the following file:

- ElfSection.hpp

4.15 iato::ElfSegment Class Reference

`#include <ElfSegment.hpp>`

Public Types

- enum **t_stype**

  the segment type

Public Member Functions

- **ElfSegment** (const t_long size, const t_octa base)
- **ElfSegment** (const t_long size, const t_octa base, **t_stype** type)
- **ElfSegment** (const int fid, void *seg, const bool mode)

4.15.1 Detailed Description

The **ElfSegment**(p. 171) class is a simple class that holds the data associated with a particular elf segment. The class cannot be constructed directly. Unlike a section, a segment does not have a name. Block segment are built from the elf file, from loadable segments.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 iato::ElfSegment::ElfSegment (const t_long *size*, const t_octa *base*)

create an elf segment by base and size

Parameters:

size the segment size

base the base address

4.15.2.2 iato::ElfSegment::ElfSegment (const t_long *size*, const t_octa *base*, t_stype *type*)

create an elf segment by base and size and type

Parameters:

size the segment size

base the base address

type the segment type

4.15.2.3 iato::ElfSegment::ElfSegment (const int *fid*, void * *seg*, const bool *mode*)

create a new elf by segment and mode

Parameters:

fid the elf file id

seg the elf segment header

mode the endian mode

The documentation for this class was generated from the following file:

- ElfSegment.hpp

4.16 iato::ElfStack Class Reference

`#include <ElfStack.hpp>`

Public Member Functions

- **ElfStack** (void)
  create a new default stack
- **ElfStack** (**ElfKernel** *ekp)
- void **setstkva** (const t_octa addr)
  set the elf stack address
- t_octa **getstkva** (void) const
  the elf stack address
- void **pargs** (void) const
  print the stack arguments after initialization

4.16.1 Detailed Description

The **ElfStack**(p. 173) class is a simple segment that represents the process stack. The stack can be initialized by default or by context. The ABI specifies that the stack grows towards lower address. Initially, the stack is created with a certain default size. After some elf initialization, the initialized stack pointer indicates the current stack pointer. Normally it should not be changed after this. The initialized stack pointer can be set to initialize the processor with the current stack pointer (sp). The stack is initialized with the read/write/execute permission.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 iato::ElfStack::ElfStack (ElfKernel * *ekp*)

create a new stack by parameters

Parameters:
ekp the elf kernel parameters

The documentation for this class was generated from the following file:

- ElfStack.hpp

4.17 iato::ElfTable Class Reference

`#include <ElfTable.hpp>`

Inherited by **iato::ElfArgs**, and **iato::ElfEnvp**.

Public Member Functions

- **ElfTable** (void)
  create an empty character table
- virtual ∼**ElfTable** (void)
  destroy this character table
- virtual void **reset** (void)
  reset this argument table
- void **add** (const string &s)
- virtual long **getargc** (void) const
  the number of arguments
- virtual long **getargs** (void) const
  the argument block size
- virtual t_byte ∗ **getargb** (void) const
  the argument block array
- virtual t_octa ∗ **getargv** (const t_octa addr) const

4.17.1 Detailed Description

The **ElfTable**(p. 174) class is a simple class that manages a vector of strings and produces a block of characters associated with a character block pointer relocated at a certain address. A typical example is the argument string for a program which is a table of string pointers. In order to operate, the string vector must be filled with the string values. The ”getargb” methods returns a character block which is padded modulo the abi alignement. The ”getargs” method returns the complete block size. The ”getargv” method returns a block of pointers relocated at a certain address. The ”getargc” method returns the number of pointers (aka the number of strings) in the argument vector. Obviously, adding a string between various getxxxx calls will result in inconsistent tables.

4.17.2 Member Function Documentation

4.17.2.1 void iato::ElfTable::add (const string & *s*)

add a new string to the table

Parameters:
s the string to add

4.17.2.2 virtual t_octa* iato::ElfTable::getargv (const t_octa *addr*) const [virtual]

relocate the argument vector at a certain address

Parameters:

addr the base address to relocate

The documentation for this class was generated from the following file:

- ElfTable.hpp

4.18 iato::ElfText Class Reference

`#include <ElfText.hpp>`

Inherits **iato::ElfMemory**.

Public Member Functions

- **ElfText** (void)
  create a default memory
- void **add** (Memory *mem)
- virtual void **addscn** (**ElfSection** *scn)
- **ElfSection** * **getscn** (const long index) const
  a section by index

4.18.1 Detailed Description

The **ElfText**(p. 176) class is a specialized elf memory object that is built from an elf file. The class acts as an array of elf sections built with text section. The associated section are those that have the text associated flag sets. Note that such section are built generally with read/execute flag only.

4.18.2 Member Function Documentation

4.18.2.1 void iato::ElfText::add (Memory * *mem*) [virtual]

add a memory only if is an elf section

Parameters:
mem the memory to add

Reimplemented from **iato::ElfMemory** (p. 167).

4.18.2.2 virtual void iato::ElfText::addscn (ElfSection * *scn*) [virtual]

add a section to this memory

Parameters:
scn the section to add

The documentation for this class was generated from the following file:

- ElfText.hpp

4.19 iato::Etx Class Reference

`#include <Etx.hpp>`

Public Member Functions

- **Etx** (void)

  create a new context

- void **reset** (void)

  reset this context

- void **update** (const t_arch arch)

  update this context with a particular architecture

4.19.1 Detailed Description

The **Etx**(p. 177) class is the micro-architecture context class. This class is derived from the isa context class and provides additional parameters that are used the elf engine. Most of these parameters are used to derive a correct process image.

The documentation for this class was generated from the following file:

- Etx.hpp

Part III

Kernel Library

Chapter 5

Kernel Library Compound Index

5.1 IATO KERNEL LIBRARY Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

Kernel Library Reference

6.1 iato::KrnExit Class Reference

`#include <KrnExit.hpp>`

Public Member Functions

- **KrnExit** (const t_octa status)
- **KrnExit** (const **KrnExit** &that)
- **KrnExit** & **operator=** (const **KrnExit** &that)
- t_octa **getstatus** (void) const

  the exit status

- void **print** (void) const

  print the exception message

6.1.1 Detailed Description

The krnExit exception is a special exception that is thrown by the interrupt engine when an exit system call has been received. The exception holds the exit status value

6.1.2 Constructor & Destructor Documentation

6.1.2.1 iato::KrnExit::KrnExit (const t_octa *status*)

create a new exit exception with a status

Parameters:

status the exception status

6.1.2.2 iato::KrnExit::KrnExit (const KrnExit & *that*)

copy construct this exception

Parameters:
that the exception to copy

6.1.3 Member Function Documentation

6.1.3.1 KrnExit& iato::KrnExit::operator= (const KrnExit & *that*)

assign an exception to this one

Parameters:
that the exception to assign

The documentation for this class was generated from the following file:

- KrnExit.hpp

6.2 iato::Syscall Class Reference

`#include <Syscall.hpp>`

Public Member Functions

- **Syscall** (void)

  create a default syscall

- **Syscall** (ElfExec ∗mem)
- **Syscall** (Rse ∗rse, Register ∗rbk, ElfExec ∗mem)
- void **setrse** (Rse ∗rse)
- void **setrbk** (Register ∗rbk)
- void **setmem** (ElfExec ∗mem)
- void **apply** (void)

  apply this syscall plugin

- void **apply** (const Interrupt &vi)

6.2.1 Detailed Description

The **Syscall**(p. 185) class is a special plugin designed to handle system calls. The plugin is constructed with the rse, the register bank and the system memory. By convention, the system call number is stored in gr[15] and the system call parameters are in the standard registers starting at gr[32]. The rse is part of this plugin in order to find the right register for a given argument. Upon completion, the return value is store in gr[10]. Generally a -1 value indicates an error. The plugin can directly work with virtual interrupt. In that case, the system expect that the immediate break value is 0x10000 as defined by the IA64 ISA. Other value rethrow the interrupt.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 iato::Syscall::Syscall (ElfExec ∗ *mem*)

create a syscall plugin with a memory

Parameters:

mem the memory image

6.2.2.2 iato::Syscall::Syscall (Rse ∗ *rse*, Register ∗ *rbk*, ElfExec ∗ *mem*)

create a plugin with a rse, register bank and memory

Parameters:

rse the register stack engine

rbk the register bank

mem the executable memory

6.2.3 Member Function Documentation

6.2.3.1 void iato::Syscall::apply (const Interrupt & *vi*)

apply this syscall plugin with an interrupt

Parameters:
vi the virtual interrupt

6.2.3.2 void iato::Syscall::setmem (ElfExec * *mem*)

set the system call memory object

Parameters:
mem the memory object

6.2.3.3 void iato::Syscall::setrbk (Register * *rbk*)

set the system call register bank object

Parameters:
rbk the register bank object

6.2.3.4 void iato::Syscall::setrse (Rse * *rse*)

set the system call rse object

Parameters:
rse the rse object

The documentation for this class was generated from the following file:

- Syscall.hpp

Part IV

MAC Library

Chapter 7

MAC Library Compound Index

7.1 IATO MAC LIBRARY Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 8

MAC Library Reference

8.1 iato::Bdb Class Reference

`#include <Bdb.hpp>`

Public Member Functions

- **Bdb** (void)
  create a default bdb
- **Bdb** (const long size)
- **Bdb** (**Mtx** *mtx)
- ~**Bdb** (void)
  destroy this bdb
- void **reset** (void)
  reset this bdb
- void **report** (void) const
  report this resource
- long **getsize** (void) const
  the bdb size
- bool **isempty** (void) const
  true if the buffer is empty
- bool **isfull** (void) const
  true if the buffer is full
- bool **isthr** (void) const
  true if the buffer is at the threshold
- void **push** (const Bundle &bndl)
- void **back** (const Bundle &bndl)

- Bundle **pop** (void)

  a bundle from the buffer

8.1.1 Detailed Description

The **Bdb**(p. 191) class is a bundle decoupling buffer. The **Bdb**(p. 191) is used primarily within an in-order machine to accumulate bundle and rotate them. Bundle are pushed on a line basis and consumed on a bundle basis. A threshold that represents the buffer size minus one line indicates that the decoupling buffer is almost full.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 iato::Bdb::Bdb (const long *size*)

create a bdb by size

Parameters:
size the bdb size

8.1.2.2 iato::Bdb::Bdb (Mtx ∗ *mtx*)

create a bdb by context

Parameters:
mtx the architectural context

8.1.3 Member Function Documentation

8.1.3.1 void iato::Bdb::back (const Bundle & *bndl*)

push back a bundle in the buffer

Parameters:
bndl the bundle to push back

8.1.3.2 void iato::Bdb::push (const Bundle & *bndl*)

push a bundle in the buffer

Parameters:
bndl the bundle to push

The documentation for this class was generated from the following file:

- Bdb.hpp

8.2 iato::Bimodal Class Reference

`#include <Bimodal.hpp>`

Inherits **iato::Branch**.

Public Member Functions

- **Bimodal** (void)

  create a default bimodal predictor

- **Bimodal** (**Mtx** ∗mtx)
- **Bimodal** (**Mtx** ∗mtx, const string &name)
- ∼**Bimodal** (void)

  destroy this predictor

- void **reset** (void)

  reset this prodictor

- void **report** (void) const

  report some resource information

- bool **istaken** (const t_octa cip) const

  true if the branch is predicted taken

- bool **ispredict** (const t_octa cip) const

  true if the branch can be predicted

- t_octa **predict** (const t_octa cip)
- void **update** (const t_octa cip, const bool btk, const t_octa nip, const t_octa hst)

8.2.1 Detailed Description

The **Bimodal**(p. 193) class is a simple bimodal branch prediction system. Given an address, a pattern history buffer is consulted to detect whether or not the branch is taken.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 iato::Bimodal::Bimodal (Mtx ∗ *mtx*)

create a new bimodal predictor by context

Parameters:

mtx the architectural context

8.2.2.2 iato::Bimodal::Bimodal (Mtx * *mtx*, const string & *name*)

create a new bimodal predictor by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.2.3 Member Function Documentation

8.2.3.1 t_octa iato::Bimodal::predict (const t_octa *cip*) [virtual]

predict the next ip from the current ip

Parameters:

cip the current instruction pointer

Reimplemented from **iato::Branch** (p. 202).

8.2.3.2 void iato::Bimodal::update (const t_octa *cip*, const bool *btk*, const t_octa *nip*, const t_octa *hst*) [virtual]

update the branch prediction with a current and next ip

Parameters:

cip the current ip to update

btk the branch taken flag

nip the next ip to use

hst the history to use

Reimplemented from **iato::Branch** (p. 202).

The documentation for this class was generated from the following file:

- Bimodal.hpp

8.3 iato::Bpe Class Reference

`#include <Bpe.hpp>`

Public Member Functions

- **Bpe** (void)
  create a default bypass element
- **Bpe** (**Mtx** ∗mtx)
- **Bpe** (**Mtx** ∗mtx, const string &name)
- ∼**Bpe** (void)
  destroy this bypass element
- void **reset** (void)
  reset this bypass element
- void **report** (void) const
  report this resource
- void **setuvr** (const long index, const Rid &rid, const Uvr &uvr)
- long **find** (const Rid &rid) const
  the index of a matching bypass element
- Rid **getrid** (const long index) const
  the rid associated with a bypass element
- Uvr **getuvr** (const long index) const
  the uvr associated with a bypass element
- void **update** (const Result &resl)
- void **clear** (const Rid &rid)

8.3.1 Detailed Description

The **Bpe**(p. 195) class is the bypass network element. It is used to model the operations of the bypass network across multiple functional units. One bypass element is used by a stage to forward the instruction results. All bypass element are merged into a global resource in order to form the bypass network. The bypass element size is determined by the maximum number of results a unit can produce.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 iato::Bpe::Bpe (Mtx ∗ *mtx*)

create a bypass network by context

Parameters:
mtx the architectural context

8.3.2.2 iato::Bpe::Bpe (Mtx * *mtx*, const string & *name*)

create a bypass network by context and name

Parameters:

mtx the architectural context

name the rat resource name

8.3.3 Member Function Documentation

8.3.3.1 void iato::Bpe::clear (const Rid & *rid*)

clear a bypass element by rid

Parameters:

rid the bpe rid to clear

8.3.3.2 void iato::Bpe::setuvr (const long *index*, const Rid & *rid*, const Uvr & *uvr*)

set a bypass value by index, rid and value

Parameters:

index the element index

rid the rid to map

uvr the uvr to map

8.3.3.3 void iato::Bpe::update (const Result & *resl*)

update the bpe with a result

Parameters:

resl the result used to update the bpe

The documentation for this class was generated from the following file:

- Bpe.hpp

8.4 iato::Bpn Class Reference

`#include <Bpn.hpp>`

Public Member Functions

- **Bpn** (void)
  create an empty network
- **Bpn** (**Mtx** ∗mtx)
- **Bpn** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)
  reset this network
- void **report** (void) const
  report this resource
- void **add** (**Bpe** ∗bpe)
- Uvr **eval** (const Rid &rid) const
- void **eval** (Operand &oprd) const
- void **predup** (**Ssi** &inst) const
- void **predup** (**Ssi** &inst, Operand &oprd) const
- void **update** (Operand &oprd) const
- void **update** (const **Ssi** &ssi, Result &resl) const
- void **clear** (const Rid &rid)
- void **clear** (const Result &resl)

8.4.1 Detailed Description

The **Bpn**(p. 197) class is the bypass network. It is a collection of bypass element. Each element is allocated by a stage that needs result forwarding. The bypass network can be used directly to evaluate some operands. In case of match, such operand value is set automatically.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 iato::Bpn::Bpn (Mtx ∗ *mtx*)

create an empty network by context

Parameters:
mtx the architectural context

8.4.2.2 iato::Bpn::Bpn (Mtx ∗ *mtx*, const string & *name*)

create an empty network by context and name

Parameters:
mtx the architectural context
name the resource name

8.4.3 Member Function Documentation

8.4.3.1 void iato::Bpn::add (Bpe * *bpe*)

add a network element

Parameters:
bpe the element to add

8.4.3.2 void iato::Bpn::clear (const Result & *resl*)

clear a bypass element by result

Parameters:
resl the result used for clearing

8.4.3.3 void iato::Bpn::clear (const Rid & *rid*)

clear a bypass element by rid

Parameters:
rid the bpe rid to clear

8.4.3.4 void iato::Bpn::eval (Operand & *oprd*) const

evaluate an operand in the network

Parameters:
oprd the operand to evaluate

8.4.3.5 Uvr iato::Bpn::eval (const Rid & *rid*) const

evaluate an rid in the bypass network

Parameters:
rid the rid to evaluate

8.4.3.6 void iato::Bpn::predup (Ssi & *inst*, Operand & *oprd*) const

update the predicate from the bypass network

Parameters:
inst the instruction to update

oprd the operand to update

8.4.3.7 void iato::Bpn::predup (Ssi & *inst*) const

update the predicate from the bypass network

Parameters:
inst the instruction to update

8.4.3.8 void iato::Bpn::update (const Ssi & *ssi*, Result & *resl*) const

update the result with rpm value

Parameters:
ssi the instruction for update
resl the result to update

8.4.3.9 void iato::Bpn::update (Operand & *oprd*) const

update an operand in the network

Parameters:
oprd the operand to evaluate

The documentation for this class was generated from the following file:

- Bpn.hpp

8.5 iato::Branch Class Reference

`#include <Branch.hpp>`

Inherited by **iato::Bimodal**, **iato::Gshare**, and **iato::Gskew**.

Public Member Functions

- **Branch** (void)

  create a default branch predictor

- **Branch** (const string &name)
- **Branch** (**Mtx** ∗mtx)
- **Branch** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)

  reset this branch system

- void **report** (void) const

  report some resource information

- virtual bool **istaken** (const t_octa addr) const

  true if the branch is predicted taken

- virtual bool **ispredict** (const t_octa addr) const

  true if the branch can be predicted

- virtual t_octa **gethist** (void) const

  the predictor history

- virtual void **sethist** (const t_octa hist)
- virtual t_octa **nextip** (const t_octa cip, const long ws)
- virtual t_octa **predict** (const t_octa cip)
- virtual void **update** (const t_octa cip, const bool btk, const t_octa nip, const t_octa hst)
- virtual void **markbr** (const Instr &inst, const Result &resl, const bool btk)

Static Public Member Functions

- **Branch** ∗ **mkbr** (**Mtx** ∗mtx)

Protected Attributes

- t_octa **d_hist**

  the predictor history

8.5.1 Detailed Description

The **Branch**(p. 200) class is a class that models a branch prediction system. Given an instruction pointer (IP) and an instruction window size, the branch prediction system computes the next instruction pointer. The default implementation is to add the instruction window size to the current ip. Complex sstem can also be devised by derivation of this class.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 iato::Branch::Branch (const string & *name*)

create a branch predictor by name

Parameters:

name the branch resource name

8.5.2.2 iato::Branch::Branch (Mtx ∗ *mtx*)

create a new branch system by context

Parameters:

mtx the architectural context

8.5.2.3 iato::Branch::Branch (Mtx ∗ *mtx*, const string & *name*)

create a new branch system by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.5.3 Member Function Documentation

8.5.3.1 virtual void iato::Branch::markbr (const Instr & *inst*, const Result & *resl*, const bool *btk*) [virtual]

update the branch system with an instruction, result and cancel flag

Parameters:

inst the branch instruction

resl the instruction result

btk the branch taken flag

8.5.3.2 Branch∗ iato::Branch::mkbr (Mtx ∗ *mtx*) [static]

return a new branch predictor by context

Parameters:

mtx the architectural context

8.5.3.3 virtual t_octa iato::Branch::nextip (const t_octa *cip*, const long *ws*) [virtual]

compute the next ip from the current ip and window size

Parameters:

cip the current instruction pointer

ws the window size

8.5.3.4 virtual t_octa iato::Branch::predict (const t_octa *cip*) [virtual]

predict the next ip from the current ip

Parameters:

cip the current instruction pointer

Reimplemented in **iato::Bimodal** (p. 194), **iato::Gshare** (p. 229), and **iato::Gskew** (p. 231).

8.5.3.5 virtual void iato::Branch::sethist (const t_octa *hist*) [virtual]

set the predictor history

Parameters:

hist the history to set

Reimplemented in **iato::Gshare** (p. 229), and **iato::Gskew** (p. 231).

8.5.3.6 virtual void iato::Branch::update (const t_octa *cip*, const bool *btk*, const t_octa *nip*, const t_octa *hst*) [virtual]

update the branch prediction with a current and next ip

Parameters:

cip the current ip to update

btk the branch taken flag

nip the next ip to use

hst the history to use

Reimplemented in **iato::Bimodal** (p. 194), **iato::Gshare** (p. 229), and **iato::Gskew** (p. 231).

The documentation for this class was generated from the following file:

- Branch.hpp

8.6 iato::Btb Class Reference

`#include <Btb.hpp>`

Public Member Functions

- **Btb** (void)

  create a default btb

- **Btb** (const long size)
- **Btb** (**Mtx** ∗mtx)
- ∼**Btb** (void)

  destroy this btb

- void **reset** (void)

  reset this btb

- long **getsize** (void) const

  the btb size

- bool **isvalid** (const t_octa addr) const

  true if the address is valid

- t_octa **gettrg** (const t_octa addr) const
- void **update** (const t_octa addr, const t_octa targ)

8.6.1 Detailed Description

The **Btb**(p. 203) class is a branch target buffer class. The btb is used to to store for a given address, the associated target address. A given btb uses a valid bit, the instruction address abd the target address.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 iato::Btb::Btb (const long *size*)

create a btb with a size

Parameters:

size the btb size

8.6.2.2 iato::Btb::Btb (Mtx ∗ *mtx*)

create a btb with a context

Parameters:

mtx the architectural context

8.6.3 Member Function Documentation

8.6.3.1 t_octa iato::Btb::gettrg (const t_octa *addr*) const

get the btb target by address

Parameters:

addr the address tag

8.6.3.2 void iato::Btb::update (const t_octa *addr*, const t_octa *targ*)

update the btb with an address and a target

Parameters:

addr the address used for update

targ the target address to store

The documentation for this class was generated from the following file:

- Btb.hpp

8.7 iato::Cache Class Reference

`#include <Cache.hpp>`

Inherited by **iato::CacheDirect**.

Public Types

- enum **t_cmtype**

  the cache types

- enum **t_cmupd**

  cache update policy

- enum **t_cmwrt**

  cache write policy

Public Member Functions

- **Cache** (void)

  create a default cache memory

- virtual void **setparam** (const long level, const **t_cmtype** type, const **t_cmupd** cupd, const **t_cmwrt** cwrt)
- virtual **t_cmtype gettype** (void) const

  the cache type

- virtual long **getlevel** (void) const

  the cache level

- virtual void **update** (const t_octa addr, const t_byte *line)=0

Protected Attributes

- **t_cmtype d_type**

  the cache type

- **t_cmupd d_cupd**

  cache update

- **t_cmwrt d_cwrt**

  cache write

- long **d_level**

  the cache level

8.7.1 Detailed Description

The **Cache**(p. 205) class is an abstract class that defines the basic interface for a cache memory subsystem. The class is derived from the memory interface and shall be used to design larger system.

8.7.2 Member Function Documentation

8.7.2.1 virtual void iato::Cache::setparam (const long *level*, const t_cmtype *type*, const t_cmupd *cupd*, const t_cmwrt *cwrt*) [virtual]

set the cache parameters

Parameters:

level the cache level

type the cache type

cupd the cache update policy

cwrt the cache write policy

8.7.2.2 virtual void iato::Cache::update (const t_octa *addr*, const t_byte * *line*) [pure virtual]

update a line with a buffer

Parameters:

addr the base address to update

line the line buffer to update

Implemented in **iato::CacheDirect** (p. 210).

The documentation for this class was generated from the following file:

- Cache.hpp

8.8 iato::CacheBlock Class Reference

`#include <CacheBlock.hpp>`

Public Member Functions

- **CacheBlock** (const long size, const long blsz)
- ~**CacheBlock** (void)

  destroy this cache block

- void **reset** (void)

  reset this cache block

- bool **isvalid** (const long index, const t_octa addr) const

  true if the line address is valid

- t_byte **readbyte** (const long index, const t_octa addr) const
- void **writebyte** (const long index, const t_octa addr, const t_byte byte)
- void **update** (const long index, const t_octa addr, const t_byte *line)

8.8.1 Detailed Description

The **CacheBlock**(p. 207) class is a simple class that implements the functionality of a cache block. A cache block is defined as an array of N entries with a valid bit, a tag and a line. For simplicity, the tag is stored as an octa with a mask associated with the block. The line is defined as a set of bytes.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 iato::CacheBlock::CacheBlock (const long *size*, const long *blsz*)

create a cache block by size and mask

Parameters:

size the block size

blsz the block line size

8.8.3 Member Function Documentation

8.8.3.1 t_byte iato::CacheBlock::readbyte (const long *index*, const t_octa *addr*) const

read a line byte at a certain address

Parameters:

index the line index in the block

addr the address to read the byte

8.8.3.2 void iato::CacheBlock::update (const long *index*, const t_octa *addr*, const t_byte * *line*)

update a line with a buffer

Parameters:

index the line index in the block

addr the base address to update

line the line buffer to update

8.8.3.3 void iato::CacheBlock::writebyte (const long *index*, const t_octa *addr*, const t_byte *byte*)

write a line byte at a certain address

Parameters:

index the line index in the block

addr the address to read the byte

byte the byte to write

The documentation for this class was generated from the following file:

- CacheBlock.hpp

8.9 iato::CacheDirect Class Reference

`#include <CacheDirect.hpp>`

Inherits **iato::Cache**.

Public Member Functions

- **CacheDirect** (const long size, const long clsz)
- ∼**CacheDirect** (void)

  destroy this cache

- void **reset** (void)

  reset this cache

- bool **isvalid** (const t_octa addr) const

  true if the address is valid

- t_byte **readbyte** (const t_octa addr) const
- void **writebyte** (const t_octa addr, const t_byte byte)
- void **update** (const t_octa addr, const t_byte ∗line)

8.9.1 Detailed Description

The **CacheDirect**(p. 209) class is cache memory class that implements the direct mapping strategy. With a direct cache, there is only one set and a unique position for a cache line. This is the simplest cache memory.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 iato::CacheDirect::CacheDirect (const long *size*, const long *clsz*)

create a direct cache by size and line size

Parameters:

size the cache size

clsz the block line size

8.9.3 Member Function Documentation

8.9.3.1 t_byte iato::CacheDirect::readbyte (const t_octa *addr*) const

read a byte at a certain address

Parameters:

addr the address to read

8.9.3.2 void iato::CacheDirect::update (const t_octa *addr*, const t_byte * *line*) [virtual]

update a line with a buffer

Parameters:

addr the base address to update

line the line buffer to update

Implements **iato::Cache** (p. 206).

8.9.3.3 void iato::CacheDirect::writebyte (const t_octa *addr*, const t_byte *byte*)

write a byte at a certain address

Parameters:

addr the address to write the byte

byte the byte to write

The documentation for this class was generated from the following file:

- CacheDirect.hpp

8.10 iato::Delayable Class Reference

`#include <Delayable.hpp>`

Inherits **iato::Runnable**.

Inherited by **iato::Stage**, and **iato::Weakable**.

Public Member Functions

- **Delayable** (void)

  create a default delayable

- **Delayable** (const string &name)
- void **reset** (void)

  reset this delayable interface

- void **flush** (void)

  flush this delayable interface

- void **run** (void)

  run this delayable object

- virtual void **activate** (void)=0

  activate this delayable object

- virtual void **setdlat** (const long dlat)
- virtual long **getdlat** (void) const

  the delayable latency

Protected Attributes

- long **d_dlat**

  the delay latency

- long **d_dcnt**

  the delay counter

8.10.1 Detailed Description

The **Delayable**(p. 211) class is a special runnable class that provides latency controlled runnable method. Upon a call to the run method, a latency counter is decreased until reaching 0. At that point the activate method is called and the counter is resetted. Unlike the runnable method, the delayable class can be used for interface that operates at different frequency that the main one, or that needs to provides delayed operation.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 iato::Delayable::Delayable (const string & *name*)

create a new delayable by name

Parameters:
name the delayable resource name

8.10.3 Member Function Documentation

8.10.3.1 virtual void iato::Delayable::setdlat (const long *dlat*) [virtual]

set the delayable latency

Parameters:
dlat the delayable latency

The documentation for this class was generated from the following file:

- Delayable.hpp

8.11 iato::Detect Class Reference

`#include <Detect.hpp>`

Public Member Functions

- **Detect** (void)

  create a default detect logic

- **Detect** (**Mtx** ∗mtx)
- **Detect** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)

  reset this detection logic

- void **report** (void) const

  report some resource information

- bool **chksip** (const **Ssi** &ssi, const Result &resl) const

  true if the ip speculation is correct

- bool **chkspp** (const **Ssi** &ssi) const

  true if the predicate prediction is valid

- void **bind** (Register ∗rbk)

8.11.1 Detailed Description

The **Detect**(p. 213) class is a special resource used to detect speculative correctness. The main operation performed by the detection logic is twofold. First, for speculative rse operation, the detection logic make sure that any cfm part of an execution result has been correctly speculated. Second, the detection logic make sure that the speculative next ip associated with branch instruction is also correct. If an error is found, the associated bit are set in the instruction record for further recovery. The detection logic is also involved in validating the predicate prediction. The speculative predicate value is checked against the real value found in the register bank. Note that with an out-of-order engine, the detection logic will operate only if the register bank is in a correct state with respect to the checked predicate.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 iato::Detect::Detect (Mtx ∗ *mtx*)

create a new detection logic by context

Parameters:

mtx the architectural context

8.11.2.2 iato::Detect::Detect (Mtx * *mtx*, const string & *name*)

create a new detection logic by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.11.3 Member Function Documentation

8.11.3.1 void iato::Detect::bind (Register * *rbk*)

bind the register bank

Parameters:

rbk the reggister bank to bind

The documentation for this class was generated from the following file:

- Detect.hpp

8.12 iato::Disperse Class Reference

`#include <Disperse.hpp>`

Public Member Functions

- **Disperse** (void)
  create a default dispersal engine
- **Disperse** (const string &name)
- **Disperse** (**Mtx** ∗mtx)
- **Disperse** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)
  reset this resource
- void **report** (void) const
  report this resource
- virtual bool **isback** (const Bundle &bndl) const
  true if a bundle must be pushed back
- virtual bool **isnext** (const Bundle &bnld) const
  true if a bundle can be followed by another one
- virtual void **expand** (Bundle &bndl, const long boix)
- virtual void **bind** (**Spb** ∗ipb, **Iib** ∗iib, **Scoreboard** ∗psb)

Protected Attributes

- **Spb** ∗ **p_ipb**
  the issue port buffer
- **Iib** ∗ **p_iib**
  the interrupt buffer
- **Scoreboard** ∗ **p_psb**
  the scoreboard

8.12.1 Detailed Description

the **Disperse**(p. 215) class is a generic class that can disperse a bundle into an issue port buffer. The class is a generic one and the simplest implementation can disperse an instruction to the first available unit only. More complex rule can be added by derivation if needed. In the base class, A type instruction are not disperse to either M or I. For that kind of dispersal, another disperser is required. The expand method disperse a bundle into the issue port buffer. If the bundle has been succesfully dispersed, the method returns true. If the bundle is partially dispersed, the dispersed instruction are marked invalid in the bundle and the method returns false.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 iato::Disperse::Disperse (const string & *name*)

create a dispersal engine by name

Parameters:
name the dispersal engine resource name

8.12.2.2 iato::Disperse::Disperse (Mtx ∗ *mtx*)

create a new dispersal engine system by context

Parameters:
mtx the architectural context

8.12.2.3 iato::Disperse::Disperse (Mtx ∗ *mtx*, const string & *name*)

create a new dispersal engine system by context and name

Parameters:
mtx the architectural context
name the dispersal engine resource name

8.12.3 Member Function Documentation

8.12.3.1 virtual void iato::Disperse::bind (Spb ∗ *ipb*, Iib ∗ *iib*, Scoreboard ∗ *psb*) [virtual]

bind the ipb and the scoreboard

Parameters:
iib the interrupt buffer
ipb the ipb to bind
psb the scoreboard to bind

8.12.3.2 virtual void iato::Disperse::expand (Bundle & *bndl*, const long *boix*) [virtual]

disperse a bundle into an issue port buffer

Parameters:
bndl the bundle to expand
boix the bundle index

The documentation for this class was generated from the following file:

- Disperse.hpp

8.13 iato::Dsi Class Reference

`#include <Dsi.hpp>`

Inherits **iato::Ssi**.

Public Member Functions

- **Dsi** (void)

  create a default dsi

- **Dsi** (const Instr &inst)
- **Dsi** (const **Dsi** &that)
- **Dsi** & **operator=** (const **Dsi** &that)
- **Dsi** & **operator=** (const Instr &that)
- void **reset** (void)

  reset this dsi

- bool **isready** (void) const

  true if the instruction is ready for selection

- void **setrdy** (const Rid &rid)
- void **setmob** (const long index)
- long **getmob** (void) const

  the mob index

- void **setsid** (const long index)
- long **getsid** (void) const

  the station entry

- void **setgcs** (const long index)
- long **getgcs** (void) const

  the gcs index

- void **setelat** (const long elat)
- long **getelat** (void) const

  the execution latency

- void **setrsch** (const bool rsch)
- bool **getrsch** (void) const

  the reschedule flag

- void **setpnrd** (const bool pnrd)
- bool **getpnrd** (void) const

  the predicate not ready flag

Static Public Member Functions

- bool **isprnm** (const Rid &rid)

  true if a register must be physically renamed

8.13.1 Detailed Description

The **Dsi**(p. 217) class is the dynamically scheduled instruction that is derived from the statically instruction class. The class holds additional information that are used in a dynamically scheduled environment. Such information includes in particular the the reorder buffer (ROB) entry.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 iato::Dsi::Dsi (const Instr & *inst*)

create a dsi from an instruction

Parameters:
inst the instruction to use

8.13.2.2 iato::Dsi::Dsi (const Dsi & *that*)

copy construct this dsi

Parameters:
that the dsi to copy

8.13.3 Member Function Documentation

8.13.3.1 Dsi& iato::Dsi::operator= (const Instr & *that*)

assign an instruction to this dsi

Parameters:
that the instruction to assign

Reimplemented from **iato::Ssi** (p. 308).

8.13.3.2 Dsi& iato::Dsi::operator= (const Dsi & *that*)

assign a dsi to this one

Parameters:
that the dsi to assign

8.13.3.3 void iato::Dsi::setelat (const long *elat*)

set the execution latency

Parameters:
elat the execution latency

8.13.3.4 void iato::Dsi::setgcs (const long *index*)

set the gcs index

Parameters:
index the gcs index

8.13.3.5 void iato::Dsi::setmob (const long *index*)

set the mob index

Parameters:
index the mob index

8.13.3.6 void iato::Dsi::setpnrd (const bool *pnrd*)

set the predicate not ready flag

Parameters:
pnrd the not ready flag

8.13.3.7 void iato::Dsi::setrdy (const Rid & *rid*)

set an instruction operand ready by rid

Parameters:
rid the rid use to mark the operand ready

8.13.3.8 void iato::Dsi::setrsch (const bool *rsch*)

set the instruction reschedule flag

Parameters:
rsch the reschedule flag to set

8.13.3.9 void iato::Dsi::setsid (const long *index*)

set the station entry

Parameters:
index the station index

The documentation for this class was generated from the following file:

- Dsi.hpp

8.14 iato::Eib Class Reference

`#include <Eib.hpp>`

Public Member Functions

- **Eib** (void)

  create a new interrupt buffer

- **Eib** (**Mtx** ∗mtx)
- void **reset** (void)

  reset this interrupt buffer

- void **report** (void) const

  report this resource

- bool **isempty** (void) const

  true if the interrupt buffer is empty

- void **push** (const Interrupt &vi)
- Interrupt **pop** (void)

  pop an interrupt from this buffer

8.14.1 Detailed Description

The **Eib**(p. 221) class is the external interrupt buffer that is implemented as a queue. External interrupts are supported by storing them in a queue and processing them as soon as possible. An external interrupt is by nature asynchronous to the underlying hardware and should not be confused with the synchronous one (aka instruction interrupt).

8.14.2 Constructor & Destructor Documentation

8.14.2.1 iato::Eib::Eib (Mtx ∗ *mtx*)

create a new interrupt buffer with a context

Parameters:

mtx the architectural context

8.14.3 Member Function Documentation

8.14.3.1 void iato::Eib::push (const Interrupt & *vi*)

push an interrupt in the buffer

Parameters:

vi the virtual interrupt to push

The documentation for this class was generated from the following file:

- Eib.hpp

8.15 iato::Eiq Class Reference

`#include <Eiq.hpp>`

Public Member Functions

- **Eiq** (void)
  create a new queue
- **Eiq** (**Mtx** *mtx)
- ~**Eiq** (void)
  destroy this queue
- void **reset** (void)
  reset this queue
- void **report** (void) const
  report this resource
- bool **isempty** (void) const
  true if the queue is empty
- bool **isfull** (void) const
  true if the queue is full
- bool **isthr** (void) const
  true if the queue has reached the threshold
- void **push** (const **Dsi** &dsi)
- **Dsi pop** (void)
  the oldest instruction from the queue

8.15.1 Detailed Description

The **Eiq**(p. 223) is the expand instruction queue. The queue operates like a fifo. The 'push' method push an instruction in the queue, while the 'pop' method remove an instruction from the queue. If the queue is empty, an invalid instruction is returned. If the queue is full the 'push' method throw an exception.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 iato::Eiq::Eiq (Mtx * *mtx*)

create a new queue with a context

Parameters:
mtx the architectural context

8.15.3 Member Function Documentation

8.15.3.1 void iato::Eiq::push (const Dsi & *dsi*)

push an instruction in the queue

Parameters:
dsi the instruction to push

The documentation for this class was generated from the following file:

- Eiq.hpp

8.16 iato::Gcs Class Reference

`#include <Gcs.hpp>`

Public Member Functions

- **Gcs** (**Mtx** ∗mtx)
- ∼**Gcs** (void)

  destroy this gcs

- void **reset** (void)

  reset this central

- void **report** (void) const

  report this resource

- void **setstc** (Stat ∗stc)
- void **settrc** (Tracer ∗tracer)
- long **add** (**Station** ∗sta)
- void **clear** (const long igcs, const long sidx)
- void **clear** (const **Dsi** &dsi)
- void **setcnl** (const Result &resl)
- void **setrdy** (const Rid &rid)
- void **setrdy** (const Result &resl)
- **Dsi setrdy** (const **Dsi** &inst)
- void **resched** (const **Dsi** &dsi)

  reschedule an instruction in a station

- void **dump** (void) const

  dump all stations contents

8.16.1 Detailed Description

The **Gcs**(p. 225) class is the Grand Central **Station**(p. 314) class, that is a collection of reservation station tables. The sole purpose of creating such resource is to permit a particular pipelane stage to broadcast ready operands to all stations.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 iato::Gcs::Gcs (Mtx ∗ *mtx*)

create an empty central with a context

Parameters:

mtx the architectural context

8.16.3 Member Function Documentation

8.16.3.1 long iato::Gcs::add (Station * *sta*)

add a station to this central and return its index

Parameters:
sta the station to add

8.16.3.2 void iato::Gcs::clear (const Dsi & *dsi*)

clear a station etry by instruction

Parameters:
dsi the instruction used to clear

8.16.3.3 void iato::Gcs::clear (const long *igcs*, const long *sidx*)

clear a station entry by gcs and station index

Parameters:
igcs the gcs index
sidx the station index

8.16.3.4 void iato::Gcs::setcnl (const Result & *resl*)

broadcast a cancel flag to all stations

Parameters:
resl the result to broadcast

8.16.3.5 Dsi iato::Gcs::setrdy (const Dsi & *inst*)

broadcast an instruction to all station or decrease latency

Parameters:
inst the instruction to check

8.16.3.6 void iato::Gcs::setrdy (const Result & *resl*)

broadcast a result to all station

Parameters:
resl the result to broadcast

8.16.3.7 void iato::Gcs::setrdy (const Rid & *rid*)

broadcast a rid to all station

Parameters:
rid the rid to broadcast

8.16.3.8 void iato::Gcs::setstc (Stat * *stc*)

set the stat collector to all stations

Parameters:
stc the stat collector to set

8.16.3.9 void iato::Gcs::settrc (Tracer * *tracer*)

bind the tracer to all stations

Parameters:
tracer the resource tracer to bind

The documentation for this class was generated from the following file:

- Gcs.hpp

8.17 iato::Gshare Class Reference

`#include <Gshare.hpp>`

Inherits **iato::Branch**.

Public Member Functions

- **Gshare** (void)
  create a default gshare predictor
- **Gshare** (**Mtx** ∗mtx)
- **Gshare** (**Mtx** ∗mtx, const string &name)
- ∼**Gshare** (void)
  destroy this predictor
- void **reset** (void)
  reset this prodictor
- void **report** (void) const
  report some resource information
- bool **istaken** (const t_octa cip) const
  true if the branch is predicted taken
- bool **ispredict** (const t_octa cip) const
  true if the branch can be predicted
- void **sethist** (const t_octa hist)
- t_octa **gethist** (void) const
  the predictor history
- t_octa **predict** (const t_octa cip)
- void **update** (const t_octa cip, const bool btk, const t_octa nip, const t_octa hst)

8.17.1 Detailed Description

The **Gshare**(p. 228) class is a gloabal history branch prediction system. Given an address, this address is combined (xor) with the gobal history register value to produce an entry into the pht. The pht value is branch prediction value.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 iato::Gshare::Gshare (Mtx ∗ *mtx*)

create a new gshare predictor by context

Parameters:

mtx the architectural context

8.17.2.2 iato::Gshare::Gshare (Mtx * *mtx*, const string & *name*)

create a new gshare predictor by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.17.3 Member Function Documentation

8.17.3.1 t_octa iato::Gshare::predict (const t_octa *cip*) [virtual]

predict the next ip from the current ip

Parameters:

cip the current instruction pointer

Reimplemented from **iato::Branch** (p. 202).

8.17.3.2 void iato::Gshare::sethist (const t_octa *hist*) [virtual]

set the predictor history

Parameters:

hist the history to set

Reimplemented from **iato::Branch** (p. 202).

8.17.3.3 void iato::Gshare::update (const t_octa *cip*, const bool *btk*, const t_octa *nip*, const t_octa *hst*) [virtual]

update the branch prediction with a current and next ip

Parameters:

cip the current ip to update

btk the branch taken flag

nip the next ip to use

hst the history to use

Reimplemented from **iato::Branch** (p. 202).

The documentation for this class was generated from the following file:

- Gshare.hpp

8.18 iato::Gskew Class Reference

`#include <Gskew.hpp>`

Inherits **iato::Branch**.

Public Member Functions

- **Gskew** (void)
  create a default gskew predictor
- **Gskew** (**Mtx** ∗mtx)
- **Gskew** (**Mtx** ∗mtx, const string &name)
- ∼**Gskew** (void)
  destroy this predictor
- void **reset** (void)
  reset this prodictor
- void **report** (void) const
  report some resource information
- bool **istaken** (const t_octa cip) const
  true if the branch is predicted taken
- bool **ispredict** (const t_octa cip) const
  true if the branch can be predicted
- void **sethist** (const t_octa hist)
- t_octa **gethist** (void) const
  the predictor history
- t_octa **predict** (const t_octa cip)
- void **update** (const t_octa cip, const bool btk, const t_octa nip, const t_octa hst)

8.18.1 Detailed Description

The **Gskew**(p. 230) class is a global history branch prediction system based on the original implementation of the skewed branch predictor designed by Andre Seznec. The predictor is augmented with a branch target buffer in order to satisfy the branch class interface. The global history is a speculative history.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 iato::Gskew::Gskew (Mtx ∗ *mtx*)

create a new gskew predictor by context

Parameters:
mtx the architectural context

8.18.2.2 iato::Gskew::Gskew (Mtx * *mtx*, const string & *name*)

create a new gskew predictor by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.18.3 Member Function Documentation

8.18.3.1 t_octa iato::Gskew::predict (const t_octa *cip*) [virtual]

predict the next ip from the current ip

Parameters:

cip the current instruction pointer

Reimplemented from **iato::Branch** (p. 202).

8.18.3.2 void iato::Gskew::sethist (const t_octa *hist*) [virtual]

set the predictor history

Parameters:

hist the history to set

Reimplemented from **iato::Branch** (p. 202).

8.18.3.3 void iato::Gskew::update (const t_octa *cip*, const bool *btk*, const t_octa *nip*, const t_octa *hst*) [virtual]

update the branch prediction with a current and next ip

Parameters:

cip the current ip to update

btk the branch taken flag

nip the next ip to use

hst the history to use

Reimplemented from **iato::Branch** (p. 202).

The documentation for this class was generated from the following file:

- Gskew.hpp

8.19 iato::Hazard Class Reference

`#include <Hazard.hpp>`

Public Member Functions

- **Hazard** (void)

  create a default dispersal engine

- **Hazard** (const string &name)
- **Hazard** (**Mtx** ∗mtx)
- **Hazard** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)

  reset this resource

- void **report** (void) const

  report this resource

- virtual bool **ishazard** (void) const

  true if there is a hazard

- virtual void **bind** (**Spb** ∗spb, **Scoreboard** ∗psb)

Protected Attributes

- **Spb** ∗ **p_spb**

  the output port buffer

- **Scoreboard** ∗ **p_psb**

  the scoreboard

8.19.1 Detailed Description

the **Hazard**(p. 232) class is a generic class that can detect hazardous condition based on a particular model. The detection logic operates with a slot port buffer and a scoreboard.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 iato::Hazard::Hazard (const string & *name*)

create a dispersal engine by name

Parameters:

name the dispersal engine resource name

8.19.2.2 iato::Hazard::Hazard (Mtx * *mtx*)

create a new dispersal engine system by context

Parameters:

mtx the architectural context

8.19.2.3 iato::Hazard::Hazard (Mtx * *mtx*, const string & *name*)

create a new dispersal engine system by context and name

Parameters:

mtx the architectural context

name the dispersal engine resource name

8.19.3 Member Function Documentation

8.19.3.1 virtual void iato::Hazard::bind (Spb * *spb*, Scoreboard * *psb*) [virtual]

bind the opb and the scoreboard

Parameters:

spb the slot port buffer to bind

psb the scoreboard to bind

The documentation for this class was generated from the following file:

- Hazard.hpp

8.20 iato::Hma Class Reference

`#include <Hma.hpp>`

Inherits **iato::Runnable**.

Public Member Functions

- **Hma** (Memory *mem)
- **Hma** (**Mtx** *mtx, Memory *mem)
- void **reset** (void)

  reset this interface

- void **flush** (void)

  flush this interface

- void **report** (void) const

  report this resource

- void **run** (void)

  run this runnable object

- **Mta** * **getmta** (void) const

  the memory adapter

Protected Attributes

- **Mta** * **p_mta**

  the mta

- Memory * **p_mem**

  the memory image

- bool **d_mbb**

  the memory bypass bit

8.20.1 Detailed Description

The **Hma**(p. 234) class is the hierarchical memory architecture. The memory model is build with or without caches. In bypass mode, no cache are installed and a direct access to the memory is perfomed via the mta. In non bypass mode, the memory access is performed via a memory request interface that is normally part of a port request.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 iato::Hma::Hma (Memory * *mem*)

create a memory architecture with a memory image

Parameters:

mem the memory image to use

8.20.2.2 iato::Hma::Hma (Mtx * *mtx*, Memory * *mem*)

create a memory interface with a context and memory image

Parameters:

mtx the architectural context

mem the memory image to use

The documentation for this class was generated from the following file:

- Hma.hpp

8.21 iato::Htr Class Reference

`#include <Htr.hpp>`

Public Member Functions

- **Htr** (void)
  create a default htr
- **Htr** (const long size)
- **Htr** (**Mtx** ∗mtx)
- **Htr** (const **Htr** &that)
- void **reset** (void)
  reset this htr
- **Htr** & **operator=** (const **Htr** &that)
- long **getsize** (void) const
  the htr size
- void **sethist** (const t_octa hist)
- t_octa **gethist** (void) const
  the masked htr value
- t_octa **update** (const bool flag)

8.21.1 Detailed Description

The **Htr**(p. 236) class is a history class. The class is designed to store the a k-bits history that is used later by a predictor to hash the address (with Ga type branch predictor). Note that the maximum size is 64 bits. The **Htr**(p. 236) class is also a good candidate for predicate predictor that is built around a global history mechanism.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 iato::Htr::Htr (const long *size*)

create a htr by size

Parameters:
size the htr size

8.21.2.2 iato::Htr::Htr (Mtx ∗ *mtx*)

create a htr by context

Parameters:
mtx the architectural context

8.21.2.3 iato::Htr::Htr (const Htr & *that*)

copy construct this htr

Parameters:
that the htr to copy

8.21.3 Member Function Documentation

8.21.3.1 Htr& iato::Htr::operator= (const Htr & *that*)

assign a htr to this one

Parameters:
that the htr to assign

8.21.3.2 void iato::Htr::sethist (const t_octa *hist*)

set the htr by value

Parameters:
hist the history to use

8.21.3.3 t_octa iato::Htr::update (const bool *flag*)

update the history by flag

Parameters:
flag the flag used for update

The documentation for this class was generated from the following file:

- Htr.hpp

8.22 iato::Iib Class Reference

`#include <Iib.hpp>`

Public Member Functions

- **Iib** (void)
  create a new buffer
- **Iib** (**Mtx** ∗mtx)
- ∼**Iib** (void)
  destroy this buffer
- void **reset** (void)
  reset this buffer
- void **report** (void) const
  report this resource
- bool **isempty** (void) const
  true if the buffer is empty
- bool **isfull** (void) const
  true if the buffer is full
- bool **isthr** (void) const
  true if the iib has reached the threshold
- bool **isvalid** (const long index) const
  true if the buffer entry is valid
- void **clear** (const long index)
  clear an iib entry by index
- long **alloc** (void)
  allocate a new iib entry
- long **alloc** (const Interrupt &vi)
- Interrupt **getintr** (const long index) const
  an interrupt by index
- void **setintr** (const long index, const Interrupt &vi)
- void **setintr** (const long index, const Interrupt &vi, const bool flag)

8.22.1 Detailed Description

The **Iib**(p. 238) is the instruction interrupt buffer. The interrupt buffer allocation is generally done when the instruction is decoded and the associated index is part of the decoded instruction.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 iato::Iib::Iib (Mtx ∗ *mtx*)

create a new buffer with a context

Parameters:
mtx the architectural context

8.22.3 Member Function Documentation

8.22.3.1 long iato::Iib::alloc (const Interrupt & *vi*)

allocate a new iib entry with an interrupt

Parameters:
vi the interrupt to set

8.22.3.2 void iato::Iib::setintr (const long *index*, const Interrupt & *vi*, const bool *flag*)

set an interrupt by index with the exec flag

Parameters:
index the iib index
vi the interrupt to set
flag the exec flag to set

8.22.3.3 void iato::Iib::setintr (const long *index*, const Interrupt & *vi*)

set an interrupt by index

Parameters:
index the iib index
vi the interrupt to set

The documentation for this class was generated from the following file:

- Iib.hpp

8.23 iato::Irb Class Reference

`#include <Irb.hpp>`

Public Member Functions

- **Irb** (void)
  create a new irb
- **Irb** (**Mtx** ∗mtx)
- ∼**Irb** (void)
  destroy this irb
- void **reset** (void)
  reset this irb
- void **report** (void) const
  report this resource
- bool **isempty** (void) const
  true if the irb is empty
- bool **isvalid** (const long index) const
  true if the irb entry is valid
- void **clear** (const long index)
  clear an irb entry by idex
- long **alloc** (const **Dsi** &inst, const Result &resl)
- **Dsi getinst** (const long index) const
  an instruction entry by index
- Result **getresl** (const long index) const
  an instruction result by index

8.23.1 Detailed Description

The **Irb**(p. 240) is the instruction result buffer. The class acts as a buffer with an index setup in the rob. Each entry is made of the instruction and the instruction result. The size of the instruction buffer is as big as the rob.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 iato::Irb::Irb (Mtx ∗ *mtx*)

create a new irb with a context

Parameters:
mtx the architectural context

8.23.3 Member Function Documentation

8.23.3.1 long iato::Irb::alloc (const Dsi & *inst*, const Result & *resl*)

allocate a new irb entry

Parameters:

inst the instruction to buffer

resl the instruction result

The documentation for this class was generated from the following file:

- Irb.hpp

8.24 iato::Mbe Class Reference

`#include <Mbe.hpp>`

Public Member Functions

- **Mbe** (void)
  create a default memory element
- void **reset** (void)
  reset this memory element
- void **seteix** (const long eix)
- long **geteix** (void) const
  the element index
- void **setmrt** (const Mrt &mrt)
- void **setmrt** (const Mrt &mrt, const long eix)
- Mrt **getmrt** (void) const
  the element mrt
- Mrt **grabmrt** (void)
  the element mrt and reset

8.24.1 Detailed Description

The **Mbe**(p. 242) class is a memory bypass element. The element is built primarily with a memory request object (mrt). An element index and an ordering flag is used to decide whether or not the element can be used in a bypass network; in order to maintain the store/load ordering.

8.24.2 Member Function Documentation

8.24.2.1 void iato::Mbe::seteix (const long *eix*)

set the element index

Parameters:
eix the element index

8.24.2.2 void iato::Mbe::setmrt (const Mrt & *mrt*, const long *eix*)

set a memory element by mrt and index

Parameters:
mrt the memory request type
eix the element index

8.24.2.3 void iato::Mbe::setmrt (const Mrt & *mrt*)

set a memory element by mrt

Parameters:
mrt the memory request type

The documentation for this class was generated from the following file:

- Mbe.hpp

8.25 iato::Mbn Class Reference

`#include <Mbn.hpp>`

Public Member Functions

- **Mbn** (void)

  create an empty network

- **Mbn** (**Mtx** ∗mtx)
- **Mbn** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)

  reset this network

- void **report** (void) const

  report this resource

- void **add** (**Mbe** ∗mbe)
- Mrt **update** (const Mrt &mrt) const
- Mrt **update** (const Mrt &mrt, const long mix) const

8.25.1 Detailed Description

The **Mbn**(p. 244) class is the memory bypass network. It is a collection of memory bypass elements. Each element is allocated by a stage that needs store forwarding. The bypass network can be used directly to evaluate a load value.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 iato::Mbn::Mbn (Mtx ∗ *mtx*)

create an empty network by context

Parameters:

mtx the architectural context

8.25.2.2 iato::Mbn::Mbn (Mtx ∗ *mtx*, const string & *name*)

create an empty network by context and name

Parameters:

mtx the architectural context

name the resource name

8.25.3 Member Function Documentation

8.25.3.1 void iato::Mbn::add (Mbe * *mbe*)

add a memory element

Parameters:
mbe the element to add

8.25.3.2 Mrt iato::Mbn::update (const Mrt & *mrt*, const long *mix*) const

update an mrt with the bypass data

Parameters:
mrt the mrt to update
mix the maximum index

8.25.3.3 Mrt iato::Mbn::update (const Mrt & *mrt*) const

update an mrt with the bypass data

Parameters:
mrt the mrt to update

The documentation for this class was generated from the following file:

- Mbn.hpp

8.26 iato::Mli Class Reference

`#include <Mli.hpp>`

Public Member Functions

- **Mli** (void)

  create a default memory logic

- **Mli** (**Mtx** ∗mtx)
- **Mli** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)

  reset this memory logic

- void **report** (void) const

  report some resource information

- void **bind** (**Mta** ∗mta, **Mob** ∗mob)
- void **preset** (const **Dsi** &inst, Result &resl)
- void **update** (const **Dsi** &inst, const Result &resl)

8.26.1 Detailed Description

The **Mli**(p. 246) class is a special resource used to perform memory operations from a result object with a memory port request. This class is combined with a memory ordering buffer (mob). With a load tranaction, the port is preset with the load request. The result object is updated during a store operation.This class acts as a logic interface.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 iato::Mli::Mli (Mtx ∗ *mtx*)

create a new memory logic by context

Parameters:

mtx the architectural context

8.26.2.2 iato::Mli::Mli (Mtx ∗ *mtx*, const string & *name*)

create a new memory logic by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.26.3 Member Function Documentation

8.26.3.1 void iato::Mli::bind (Mta * *mta*, Mob * *mob*)

bind a memory request port

Parameters:

mta the memory transaction adapter

mob the memory ordering buffer

8.26.3.2 void iato::Mli::preset (const Dsi & *inst*, Result & *resl*)

preset a memory operation with a result

Parameters:

inst the instruction to process

resl the result used for request

8.26.3.3 void iato::Mli::update (const Dsi & *inst*, const Result & *resl*)

update a memory operation with a result

Parameters:

inst the instruction to process

resl the result used for request

The documentation for this class was generated from the following file:

- Mli.hpp

8.27 iato::Mob Class Reference

`#include <Mob.hpp>`

Public Member Functions

- **Mob** (void)

  create a new mob

- **Mob** (**Mtx** *mtx)
- ~**Mob** (void)

  destroy this mob

- void **reset** (void)

  reset this mob

- void **report** (void) const

  report this resource

- bool **isempty** (void) const

  true if the mob is empty

- bool **isthr** (void) const

  true if the mob has reached the threshold

- bool **isvalid** (const long index) const

  true if the mob entry is valid

- bool **iscancel** (const long index) const

  true if the mob entry has been cancelled

- void **clear** (const long index)

  clear an mob entry by index

- long **alloc** (const bool ildb, const bool istb)
- void **preset** (const long index, const Mrt &mrt)
- void **update** (const long index, const Mrt &mrt)
- void **process** (const long index, const Mrt &mrt)

8.27.1 Detailed Description

The **Mob**(p. 248) class is the memory ordering buffer. It is a rotating resource that maintain the load/store ordering. An entry is made with a valid bit, a size (aka an address mask) and an address. This mob does not hold the associated value due to the cache interference.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 iato::Mob::Mob (Mtx * *mtx*)

create a new mob with a context

Parameters:
mtx the architectural context

8.27.3 Member Function Documentation

8.27.3.1 long iato::Mob::alloc (const bool *ildb*, const bool *istb*)

allocate a new mob entry

Parameters:
ildb the instruction load flag
istb the instruction store flag

8.27.3.2 void iato::Mob::preset (const long *index*, const Mrt & *mrt*)

preset a load by index and mrt

Parameters:
index the mob store index
mrt the memory request type

8.27.3.3 void iato::Mob::process (const long *index*, const Mrt & *mrt*)

process a memory request by index and mrt

Parameters:
index the mob store index
mrt the memory request type

8.27.3.4 void iato::Mob::update (const long *index*, const Mrt & *mrt*)

commit a store by index and mrt

Parameters:
index the mob store index
mrt the memory request type

The documentation for this class was generated from the following file:

- Mob.hpp

8.28 iato::Mpr Class Reference

`#include <Mpr.hpp>`

Public Member Functions

- **Mpr** (void)
  create a default port
- **Mpr** (**Mtx** ∗mtx)
- **Mpr** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)
  reset this port mta
- void **flush** (void)
  flush this port mta
- void **report** (void) const
  report this port mta
- virtual bool **isbusy** (void) const
  the busy bit
- virtual bool **istack** (void) const
  the transaction acknowldge bit
- virtual Mrt **getmrt** (void) const
  the memory request type
- virtual Bundle **getbndl** (const long index) const
  a transaction bundle by index
- virtual void **request** (const Mrt &mrt)
- virtual void **preset** (const Mrt &mrt)
- virtual void **update** (const Mrt &mrt)
- virtual void **bind** (**Mta** ∗mta)

8.28.1 Detailed Description

The **Mpr**(p. 250) class is a memory request port. The port is bound with a memory transaction adapter (mta). If the port is associated with a memory request interface (mri), the port can be operating in blocking mode. Without mri, the port operates in direct mode (i.e without delay).

8.28.2 Constructor & Destructor Documentation

8.28.2.1 iato::Mpr::Mpr (Mtx ∗ *mtx*)

create a port by context

Parameters:
mtx the architectural context

8.28.2.2 iato::Mpr::Mpr (Mtx * *mtx*, const string & *name*)

create a port by context and name

Parameters:
mtx the architectural context
name the port name

8.28.3 Member Function Documentation

8.28.3.1 virtual void iato::Mpr::bind (Mta * *mta*) [virtual]

bind the port resources

Parameters:
mta the mta to bind

8.28.3.2 virtual void iato::Mpr::preset (const Mrt & *mrt*) [virtual]

issue a port load request by mrt

Parameters:
mrt the memory request to process

8.28.3.3 virtual void iato::Mpr::request (const Mrt & *mrt*) [virtual]

issue a port request by mrt

Parameters:
mrt the memory request to process

8.28.3.4 virtual void iato::Mpr::update (const Mrt & *mrt*) [virtual]

issue a port store request by mrt

Parameters:
mrt the memory request to process

The documentation for this class was generated from the following file:

- Mpr.hpp

8.29 iato::Msi Class Reference

`#include <Msi.hpp>`

Public Member Functions

- **Msi** (void)

  create a default memory synchro logic

- **Msi** (**Mtx** ∗mtx)
- **Msi** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)

  reset this memory logic

- void **report** (void) const

  report some resource information

- void **bind** (**Mta** ∗mta)
- void **preset** (const **Ssi** &inst, Result &resl)
- void **update** (const **Ssi** &inst, const Result &resl)

8.29.1 Detailed Description

The **Msi**(p. 252) class is a special resource used to perform in-order memory operations. The class operates with a memory port and combines a memory bypass network that can be used for subsequent load.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 iato::Msi::Msi (Mtx ∗ *mtx*)

create a new memory synchro logic by context

Parameters:

mtx the architectural context

8.29.2.2 iato::Msi::Msi (Mtx ∗ *mtx*, const string & *name*)

create a new memory logic by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.29.3 Member Function Documentation

8.29.3.1 void iato::Msi::bind (Mta * *mta*)

bind a memory request port

Parameters:

mta the memory transaction adapter

8.29.3.2 void iato::Msi::preset (const Ssi & *inst*, Result & *resl*)

preset a memory operation with a result

Parameters:

inst the instruction to process

resl the result used for request

8.29.3.3 void iato::Msi::update (const Ssi & *inst*, const Result & *resl*)

update a memory operation with a result

Parameters:

inst the instruction to process

resl the result used for request

The documentation for this class was generated from the following file:

- Msi.hpp

8.30 iato::Mta Class Reference

`#include <Mta.hpp>`

Public Types

- enum **t_tmem**

  the mta memory types

Public Member Functions

- **Mta** (void)

  create a default mta

- **Mta** (**Mtx** ∗mtx)
- **Mta** (**Mtx** ∗mtx, const string &name)
- ∼**Mta** (void)

  destroy this mta

- void **reset** (void)

  reset this mta

- void **bind** (Memory ∗mem)
- void **bind** (**t_tmem** type, Memory ∗mem)
- void **process** (Mrt &mrt)
- void **update** (const t_octa bip) const
- Bundle **getbndl** (const long index) const

  the mta bundle by index

8.30.1 Detailed Description

The **Mta**(p. 254) class is a memory transaction adapter class that adapts the memory transaction to with a result object. The adapter supports also bundle related transactions. Additionally, the adapter is designed to respond to alat request.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 iato::Mta::Mta (Mtx ∗ *mtx*)

create a mta with a context

Parameters:

mtx the architectural context

8.30.2.2 iato::Mta::Mta (Mtx * *mtx*, const string & *name*)

create a mta with a context and a name

Parameters:

mtx the architectural context

name the resource name

8.30.3 Member Function Documentation

8.30.3.1 void iato::Mta::bind (t_tmem *type*, Memory * *mem*)

bind a memory to the mta by type

Parameters:

type the memory type to bind

mem the memory to bind

8.30.3.2 void iato::Mta::bind (Memory * *mem*)

bind a memory to the mta

Parameters:

mem the memory to bind

8.30.3.3 void iato::Mta::process (Mrt & *mrt*)

process a memory request

Parameters:

mrt the memory request to process

8.30.3.4 void iato::Mta::update (const t_octa *bip*) const

update a bundle array at a certain address

Parameters:

bip the bunlde ip to use

The documentation for this class was generated from the following file:

- Mta.hpp

8.31 iato::Mtx Class Reference

`#include <Mtx.hpp>`

Public Member Functions

- **Mtx** (void)
  create a new context
- void **reset** (void)
  reset this context
- void **update** (const t_arch arch)
- long **getsbsz** (void) const
  the system bus size
- long **getsblt** (void) const
  the system bus latency
- long **gettusz** (void) const
  the total number of units

8.31.1 Detailed Description

The **Mtx**(p. 256) class is the micro-architecture context class. This class is derived from the isa context class and provides additional parameters that are used by the micro-architectural implementation.

8.31.2 Member Function Documentation

8.31.2.1 void iato::Mtx::update (const t_arch *arch*)

update this contect with a particular archietcture

Parameters:
arch the archietcture used for update

The documentation for this class was generated from the following file:

- Mtx.hpp

8.32 iato::Pbmodal Class Reference

`#include <Pbmodal.hpp>`

Inherits **iato::Predicate**.

Public Member Functions

- **Pbmodal** (void)

  create a default predictor

- **Pbmodal** (**Mtx** ∗mtx)
- **Pbmodal** (**Mtx** ∗mtx, const string &name)
- ∼**Pbmodal** (void)

  destroy this predictor

- void **reset** (void)

  reset this predictor

- void **report** (void) const

  report some resource information

- bool **isvalid** (const t_octa ip, const long slot, const long pred) const

  true if the predicate can be predicted

- bool **compute** (const t_octa ip, const long slot, const long pred) const
- void **update** (const t_octa ip, const long slot, const long pred, const bool pval, const t_octa phst)

8.32.1 Detailed Description

The **Pbmodal**(p. 257) class is a simple predicate prediction system. Given a predicate number, the pht table is accesed and a prediction is returned. This system does not use the instruction address or slot. The term 'pbmodal' is cooked from 'bimodal' with the B (branch) replaced with a P (predicate).

8.32.2 Constructor & Destructor Documentation

8.32.2.1 iato::Pbmodal::Pbmodal (Mtx ∗ *mtx*)

create a new predictor by context

Parameters:

mtx the architectural context

8.32.2.2 iato::Pbmodal::Pbmodal (Mtx * *mtx*, const string & *name*)

create a new predictor by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.32.3 Member Function Documentation

8.32.3.1 bool iato::Pbmodal::compute (const t_octa *ip*, const long *slot*, const long *pred*) const [virtual]

compute the predicate value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate number

Reimplemented from **iato::Predicate** (p. 272).

8.32.3.2 void iato::Pbmodal::update (const t_octa *ip*, const long *slot*, const long *pred*, const bool *pval*, const t_octa *phst*) [virtual]

update the predicate prediction by index and value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate index

pval the predicate value

phst the predictor history

Reimplemented from **iato::Predicate** (p. 273).

The documentation for this class was generated from the following file:

- Pbmodal.hpp

8.33 iato::Pforce Class Reference

`#include <Pforce.hpp>`

Inherits **iato::Predicate**.

Public Member Functions

- **Pforce** (void)
  create a default pforce predictor
- **Pforce** (**Mtx** ∗mtx)
- **Pforce** (**Mtx** ∗mtx, const string &name)
- void **report** (void) const
  report some resource information
- bool **isvalid** (const t_octa ip, const long slot, const long pred) const
  true if the predicate can be predicted
- bool **compute** (const t_octa ip, const long slot, const long pred) const

8.33.1 Detailed Description

The **Pforce**(p. 259) class is a simple predicate prediction system. The predicate is always predicted to a boolean value. The update is ignored.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 iato::Pforce::Pforce (Mtx ∗ *mtx*)

create a new pforce predictor by context

Parameters:
mtx the architectural context

8.33.2.2 iato::Pforce::Pforce (Mtx ∗ *mtx*, const string & *name*)

create a new pforce predictor by context and name

Parameters:
mtx the architectural context
name the branch resource name

8.33.3 Member Function Documentation

8.33.3.1 bool iato::Pforce::compute (const t_octa *ip*, const long *slot*, const long *pred*) const [virtual]

compute the predicate value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate number

Reimplemented from **iato::Predicate** (p. 272).

The documentation for this class was generated from the following file:

- Pforce.hpp

8.34 iato::Pht Class Reference

`#include <Pht.hpp>`

Public Member Functions

- **Pht** (void)
  create a default pht
- **Pht** (const long size)
- **Pht** (**Mtx** ∗mtx)
- ∼**Pht** (void)
  destroy this pht
- void **reset** (void)
  reset this pht
- long **getsize** (void) const
  the pht size
- virtual long **hash** (const t_octa addr) const
  a hashed index by address
- virtual bool **isstrong** (const long index) const
  true if the prediction is strong by index
- virtual bool **isstrong** (const t_octa addr) const
  true if the prediction is strong by address
- virtual bool **isweak** (const long index) const
  true if the prediction is weak by index
- virtual bool **isweak** (const t_octa addr) const
  true if the prediction is weak by address
- virtual bool **istrue** (const long index) const
  true if the prediction is true by index
- virtual bool **istrue** (const t_octa addr) const
  true if the prediction is true by address
- virtual void **update** (const long index, const bool flag)
- virtual void **update** (const t_octa addr, const bool flag)

8.34.1 Detailed Description

The **Pht**(p. 261) class is a pattern history table class. The class is defined as an array of 2 bits saturating counters (**Sct**(p. 299)). Given an index the pht can be updated or queried to get a prediction status.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 iato::Pht::Pht (const long *size*)

create a pht with a size

Parameters:
size the pht size

8.34.2.2 iato::Pht::Pht (Mtx * *mtx*)

create a pht with a context

Parameters:
mtx the architectural context

8.34.3 Member Function Documentation

8.34.3.1 virtual void iato::Pht::update (const t_octa *addr*, const bool *flag*) [virtual]

update the pht by address

Parameters:
addr the updating address
flag the flag used for update

8.34.3.2 virtual void iato::Pht::update (const long *index*, const bool *flag*) [virtual]

update the pht by index

Parameters:
index the pht index
flag the flag used for update

The documentation for this class was generated from the following file:

- Pht.hpp

8.35 iato::Pimodal Class Reference

`#include <Pimodal.hpp>`

Inherits **iato::Predicate**.

Public Member Functions

- **Pimodal** (void)

  create a default pimodal predictor

- **Pimodal** (**Mtx** ∗mtx)
- **Pimodal** (**Mtx** ∗mtx, const string &name)
- ∼**Pimodal** (void)

  destroy this predictor

- void **reset** (void)

  reset this predictor

- void **report** (void) const

  report some resource information

- bool **isvalid** (const t_octa ip, const long slot, const long pred) const

  true if the predicate can be predicted

- bool **compute** (const t_octa ip, const long slot, const long pred) const
- void **update** (const t_octa ip, const long slot, const long pred, const bool pval, const t_octa phst)

8.35.1 Detailed Description

The **Pimodal**(p. 263) class is a simple predicate prediction system. Given a predicate number, the pht table is accesed and a prediction is returned. This system does not use the instruction address or slot. The term 'pimodal' is cooked from 'bimodal' with the B (branch) replaced with a P (predicate).

8.35.2 Constructor & Destructor Documentation

8.35.2.1 iato::Pimodal::Pimodal (Mtx ∗ *mtx*)

create a new pimodal predictor by context

Parameters:

mtx the architectural context

8.35.2.2 iato::Pimodal::Pimodal (Mtx * *mtx*, const string & *name*)

create a new pimodal predictor by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.35.3 Member Function Documentation

8.35.3.1 bool iato::Pimodal::compute (const t_octa *ip*, const long *slot*, const long *pred*) const [virtual]

compute the predicate value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate number

Reimplemented from **iato::Predicate** (p. 272).

8.35.3.2 void iato::Pimodal::update (const t_octa *ip*, const long *slot*, const long *pred*, const bool *pval*, const t_octa *phst*) [virtual]

update the predicate prediction by index and value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate index

pval the predicate value

phst the predictor history

Reimplemented from **iato::Predicate** (p. 273).

The documentation for this class was generated from the following file:

- Pimodal.hpp

8.36 iato::Pipelane Class Reference

`#include <Pipelane.hpp>`

Inherits **iato::Stage**.

Public Member Functions

- **Pipelane** (**Mtx** ∗mtx)
- **Pipelane** (**Mtx** ∗mtx, const bool mode)
- **Pipelane** (**Mtx** ∗mtx, const string &name)
- ∼**Pipelane** (void)

  destroy this pipelane

- void **reset** (void)

  reset this pipelane

- void **flush** (void)

  flush this pipelane

- bool **isholding** (void) const

  return true if one stage is holding

- void **activate** (void)

  activate all pipelines

- void **report** (void) const

  report on all pipelane stages

- void **setstc** (Stat ∗stc)
- void **settrc** (Tracer ∗tracer)
- void **setmode** (const bool mode)
- bool **getmode** (void) const

  the blocking mode

- virtual void **clear** (void)

  clear this pipelane

- virtual long **depth** (void) const

  the pipelane maximum depth

- virtual void **add** (**Pipeline** ∗pipe)
- virtual void **setclog** (const long index, **Runnable** ∗pclog)
- void **bind** (Env ∗env, **Stage** ∗pstg, **Stage** ∗nstg)

8.36.1 Detailed Description

The **Pipelane**(p. 265) class is a collection of micro pipeline that operates in parallel. At each cycle, all parallel stages are run in parallel. Each micro pipeline do not need to have the same depth. by default, the pipelane operates in non-blocking mode. This means that if a stage is holding, only those previous stage in the same micro-pipeline will be stalled. If the pipelane operates in blocking mode, a holding stage causes all other stages to block. Such mode is used with an in-order machine.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 iato::Pipelane::Pipelane (Mtx * *mtx*)

create a default pipelane with a context

Parameters:

mtx the architectural context

8.36.2.2 iato::Pipelane::Pipelane (Mtx * *mtx*, const bool *mode*)

create a pipelane with a context and a mode

Parameters:

mtx the architectural context

mode the blocking mode

8.36.2.3 iato::Pipelane::Pipelane (Mtx * *mtx*, const string & *name*)

create a new pipelane with a context and name

Parameters:

mtx the architectural context

name the pipelane name

8.36.3 Member Function Documentation

8.36.3.1 virtual void iato::Pipelane::add (Pipeline * *pipe*) [virtual]

add a new pipeline in this pipelane

Parameters:

pipe the pipeline to add

8.36.3.2 void iato::Pipelane::bind (Env * *env*, Stage * *pstg*, Stage * *nstg*) [virtual]

bind this pipelane with an execution environment

Parameters:

env the execution environment

pstg the previous stage

nstg the next stage

Reimplemented from **iato::Stage** (p. 312).

8.36.3.3 virtual void iato::Pipelane::setclog (const long *index*, Runnable * *pclog*) [virtual]

set the pipeline control logic by index

Parameters:

index the pipeline index

pclog the pipeline control logic

8.36.3.4 void iato::Pipelane::setmode (const bool *mode*)

set the blocking mode

Parameters:

mode the mode to set

8.36.3.5 void iato::Pipelane::setstc (Stat * *stc*)

set the stat collector to all pilelines

Parameters:

stc the stat collector to set

8.36.3.6 void iato::Pipelane::settrc (Tracer * *tracer*)

set the resource tracer to all pipelines

Parameters:

tracer the resource tracer to bind

The documentation for this class was generated from the following file:

- Pipelane.hpp

8.37 iato::Pipeline Class Reference

`#include <Pipeline.hpp>`

Inherits **iato::Runnable**.

Public Member Functions

- **Pipeline** (**Mtx** ∗mtx)
- **Pipeline** (**Mtx** ∗mtx, const string &name)
- ∼**Pipeline** (void)

  destroy this pipeline

- void **reset** (void)

  reset this pipeline

- void **flush** (void)

  flush this pipeline

- bool **isholding** (const long index) const

  return true if one stage is holding

- void **run** (void)

  run all pipeline stages

- void **run** (const long index)
- void **activate** (const long index)
- void **report** (void) const

  report on all pipeline stages

- void **setstc** (Stat ∗stc)
- void **settrc** (Tracer ∗tracer)
- virtual void **clear** (void)

  clear this pipeline

- virtual long **depth** (void) const

  the pipeline depth

- virtual void **add** (**Stage** ∗stg)
- virtual **Stage** ∗ **get** (const string &name) const

  a pipeline stage by name

- virtual void **bind** (Env ∗env, **Stage** ∗pstg, **Stage** ∗nstg)
- virtual bool **ishalted** (void) const

  true if the pipeline is halted

8.37.1 Detailed Description

The **Pipeline**(p. 268) class is a generic container class that holds the pipeline stages of the selected micro-architecture. Simply speaking, a pipeline is a vector of stages that would be run sequentially. When adding a new stage, the pipeline stage is added at the end of the vector. Once created, the pipeline is connected with the ”bind” method. Running the pipeline is done by running all stages in reverse order. Normal order is done for flushing and reseting.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 iato::Pipeline::Pipeline (Mtx ∗ *mtx*)

create a default pipeline

Parameters:
mtx the architectural context

8.37.2.2 iato::Pipeline::Pipeline (Mtx ∗ *mtx*, const string & *name*)

create a new pipeline with a context and name

Parameters:
mtx the architectural context
name the pipeline name

8.37.3 Member Function Documentation

8.37.3.1 void iato::Pipeline::activate (const long *index*)

activate one pipeline stage by index

Parameters:
index the stage index to activate

8.37.3.2 virtual void iato::Pipeline::add (Stage ∗ *stg*) [virtual]

add a new stage in this pipeline

Parameters:
stg the stage to add

8.37.3.3 virtual void iato::Pipeline::bind (Env ∗ *env*, Stage ∗ *pstg*, Stage ∗ *nstg*) [virtual]

bind this pipeline with an environment and the enclosing stages

Parameters:
env the execution environment

pstg the previous stage

nstg the next stage

8.37.3.4 void iato::Pipeline::run (const long *index*)

run one pipeline stage by index

Parameters:

index the stage index to run

8.37.3.5 void iato::Pipeline::setstc (Stat * *stc*)

set the stat collector to all stages

Parameters:

stc the stat collector to set

8.37.3.6 void iato::Pipeline::settrc (Tracer * *tracer*)

set the resource tracer to all resources

Parameters:

tracer the resource tracer to bind

The documentation for this class was generated from the following file:

- Pipeline.hpp

8.38 iato::Predicate Class Reference

`#include <Predicate.hpp>`

Inherited by **iato::Pbmodal**, **iato::Pforce**, **iato::Pimodal**, **iato::Pshare**, and **iato::Pskew**.

Public Member Functions

- **Predicate** (void)
  create a default predictor
- **Predicate** (const string &name)
- **Predicate** (**Mtx** ∗mtx)
- **Predicate** (**Mtx** ∗mtx, const string &name)
- void **reset** (void)
  reset this predictor
- void **report** (void) const
  report some resource information
- virtual string **gettype** (void) const
  the predictor type
- virtual bool **isvalid** (const t_octa ip, const long slot, const long pred) const
  true if the predicate can be predicted
- virtual bool **ispredict** (const Instr &inst) const
  true if the instruction predicate can be predicted
- virtual t_octa **getphst** (void) const
  the predictor history
- virtual void **setphst** (const t_octa phst)
- virtual bool **compute** (const t_octa ip, const long slot, const long pred) const
- virtual bool **predict** (const Instr &inst) const
- virtual void **update** (const t_octa ip, const long slot, const long pred, const bool pval, const t_octa phst)
- virtual void **markpp** (const Instr &inst, const bool pval)
- virtual void **markpp** (const Instr &inst, const Result &resl)

Static Public Member Functions

- **Predicate** ∗ **mkpr** (const string &name)
  a new predicate predictor by name
- **Predicate** ∗ **mkpr** (**Mtx** ∗mtx)
  a new predicate predictor by context
- **Predicate** ∗ **mkpr** (const string &name, **Mtx** ∗mtx)
  a new predicate predictor by name and context

Protected Attributes

- string **d_type**
  the predictor type
- t_octa **d_phst**
  the predictor history

8.38.1 Detailed Description

The **Predicate**(p. 271) class is a class that models a predicate prediction system. This class is a base class that provides three types of services, namely prediction confidence (i.e avaibility), predicate prediction and system update. A complex prediction system can be built by derivation. This system, as a base system never provide prediction.

8.38.2 Constructor & Destructor Documentation

8.38.2.1 iato::Predicate::Predicate (const string & *name*)

create a predictor by name

Parameters:
name the resource name

8.38.2.2 iato::Predicate::Predicate (Mtx * *mtx*)

create a new predicate system by context

Parameters:
mtx the architectural context

8.38.2.3 iato::Predicate::Predicate (Mtx * *mtx*, const string & *name*)

create a new predicate system by context and name

Parameters:
mtx the architectural context
name the predicate resource name

8.38.3 Member Function Documentation

8.38.3.1 virtual bool iato::Predicate::compute (const t_octa *ip*, const long *slot*, const long *pred*) const [virtual]

compute the predicate value

Parameters:
ip the instruction ip

slot the instruction slot

pred the predicate number

Reimplemented in **iato::Pbmodal** (p. 258), **iato::Pforce** (p. 259), **iato::Pimodal** (p. 264), **iato::Pshare** (p. 276), and **iato::Pskew** (p. 278).

8.38.3.2 virtual void iato::Predicate::markpp (const Instr & *inst*, const Result & *resl*) [virtual]

update the predicate prediction by instruction and result

Parameters:

inst the instruction used for update

resl the result used for update

8.38.3.3 virtual void iato::Predicate::markpp (const Instr & *inst*, const bool *pval*) [virtual]

update the predicate prediction by instruction and value

Parameters:

inst the instruction used for update

pval the predicate value

8.38.3.4 virtual bool iato::Predicate::predict (const Instr & *inst*) const [virtual]

predict the instruction predicate value

Parameters:

inst the instruction used for prediction

8.38.3.5 virtual void iato::Predicate::setphst (const t_octa *phst*) [virtual]

set the predictor history

Parameters:

phst the history to set

Reimplemented in **iato::Pshare** (p. 276), and **iato::Pskew** (p. 278).

8.38.3.6 virtual void iato::Predicate::update (const t_octa *ip*, const long *slot*, const long *pred*, const bool *pval*, const t_octa *phst*) [virtual]

update the predicate prediction by index and value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate index

pval the predicate value

phst the predictor history

Reimplemented in **iato::Pbmodal** (p. 258), **iato::Pimodal** (p. 264), **iato::Pshare** (p. 276), and **iato::Pskew** (p. 278).

The documentation for this class was generated from the following file:

- Predicate.hpp

8.39 iato::Pshare Class Reference

`#include <Pshare.hpp>`

Inherits **iato::Predicate**.

Public Member Functions

- **Pshare** (void)

  create a default pshare predictor

- **Pshare** (**Mtx** ∗mtx)
- **Pshare** (**Mtx** ∗mtx, const string &name)
- ∼**Pshare** (void)

  destroy this predictor

- void **reset** (void)

  reset this predictor

- void **report** (void) const

  report some resource information

- bool **isvalid** (const t_octa ip, const long slot, const long pred) const

  true if the predicate can be predicted

- t_octa **getphst** (void) const

  the predictor history

- void **setphst** (const t_octa phst)
- bool **compute** (const t_octa ip, const long slot, const long pred) const
- void **update** (const t_octa ip, const long slot, const long pred, const bool pval, const t_octa phst)

8.39.1 Detailed Description

The **Pshare**(p. 275) class is a global history predicate prediction system. Given an address, this address is combined (xor) with the global history register value (and eventually the predicate number) to produce an entry into the pht. The pht value is the predicate prediction value.

8.39.2 Constructor & Destructor Documentation

8.39.2.1 iato::Pshare::Pshare (Mtx ∗ *mtx*)

create a new pshare predictor by context

Parameters:

mtx the architectural context

8.39.2.2 iato::Pshare::Pshare (Mtx * *mtx*, const string & *name*)

create a new pshare predictor by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.39.3 Member Function Documentation

8.39.3.1 bool iato::Pshare::compute (const t_octa *ip*, const long *slot*, const long *pred*) const [virtual]

compute the predicate value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate number

Reimplemented from **iato::Predicate** (p. 272).

8.39.3.2 void iato::Pshare::setphst (const t_octa *phst*) [virtual]

set the predictor history

Parameters:

phst the history to set

Reimplemented from **iato::Predicate** (p. 273).

8.39.3.3 void iato::Pshare::update (const t_octa *ip*, const long *slot*, const long *pred*, const bool *pval*, const t_octa *phst*) [virtual]

update the predicate prediction by index and value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate index

pval the predicate value

phst the predictor history

Reimplemented from **iato::Predicate** (p. 273).

The documentation for this class was generated from the following file:

- Pshare.hpp

8.40 iato::Pskew Class Reference

`#include <Pskew.hpp>`

Inherits **iato::Predicate**.

Public Member Functions

- **Pskew** (void)

  create a default pshare predictor

- **Pskew** (**Mtx** ∗mtx)
- **Pskew** (**Mtx** ∗mtx, const string &name)
- ∼**Pskew** (void)

  destroy this predictor

- void **reset** (void)

  reset this predictor

- void **report** (void) const

  report some resource information

- bool **isvalid** (const t_octa ip, const long slot, const long pred) const

  true if the predicate can be predicted

- t_octa **getphst** (void) const

  the predictor history

- void **setphst** (const t_octa phst)
- bool **compute** (const t_octa ip, const long slot, const long pred) const
- void **update** (const t_octa ip, const long slot, const long pred, const bool pval, const t_octa phst)

8.40.1 Detailed Description

The **Pskew**(p. 277) class is a global history predicate prediction system. Given an address, this address is combined (xor) with the global history register value (and eventually the predicate number) to produce an entry into the pht. The pht value is the predicate prediction value.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 iato::Pskew::Pskew (Mtx ∗ *mtx*)

create a new pshare predictor by context

Parameters:

mtx the architectural context

8.40.2.2 iato::Pskew::Pskew (Mtx * *mtx*, const string & *name*)

create a new pshare predictor by context and name

Parameters:

mtx the architectural context

name the branch resource name

8.40.3 Member Function Documentation

8.40.3.1 bool iato::Pskew::compute (const t_octa *ip*, const long *slot*, const long *pred*) const [virtual]

compute the predicate value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate number

Reimplemented from **iato::Predicate** (p. 272).

8.40.3.2 void iato::Pskew::setphst (const t_octa *phst*) [virtual]

set the predictor history

Parameters:

phst the history to set

Reimplemented from **iato::Predicate** (p. 273).

8.40.3.3 void iato::Pskew::update (const t_octa *ip*, const long *slot*, const long *pred*, const bool *pval*, const t_octa *phst*) [virtual]

update the predicate prediction by index and value

Parameters:

ip the instruction ip

slot the instruction slot

pred the predicate index

pval the predicate value

phst the predictor history

Reimplemented from **iato::Predicate** (p. 273).

The documentation for this class was generated from the following file:

- Pskew.hpp

8.41 iato::Rat Class Reference

`#include <Rat.hpp>`

Public Member Functions

- **Rat** (void)

  create a default rat

- **Rat** (**Mtx** ∗mtx)
- **Rat** (**Mtx** ∗mtx, const string &name)
- ∼**Rat** (void)

  destroy this rat

- void **reset** (void)

  reset this rat

- void **report** (void) const

  report this resource

- long **getsize** (t_lreg lreg) const

  the number of registers by type

- long **getmap** (t_lreg lreg, const long rnum) const
- long **getmap** (const Rid &rid) const
- long **setmap** (t_lreg lreg, const long rnum, const long mnum)
- long **setmap** (const Rid &rid, const long mnum)

8.41.1 Detailed Description

The **Rat**(p. 279) class is the ram implementation of the Register Alias Table The class establishes a binding between a logical register number and another one. For each register types, a rat is created those size is defined by the isa. A special case is induced by the gr bank, that can have more registers as defined by the isa.

8.41.2 Constructor & Destructor Documentation

8.41.2.1 iato::Rat::Rat (Mtx ∗ *mtx*)

create a new rat by context

Parameters:

mtx the architectural context

8.41.2.2 iato::Rat::Rat (Mtx * *mtx*, const string & *name*)

create a new rat by context and name

Parameters:

mtx the architectural context

name the rat resource name

8.41.3 Member Function Documentation

8.41.3.1 long iato::Rat::getmap (const Rid & *rid*) const

get the register mapping by rid

Parameters:

rid the register id to use

8.41.3.2 long iato::Rat::getmap (t_lreg *lreg*, const long *rnum*) const

get the register mapping by register type and id

Parameters:

lreg the register type

rnum the register number

8.41.3.3 long iato::Rat::setmap (const Rid & *rid*, const long *mnum*)

set the register mapping by rid and return the old one

Parameters:

rid the register id to set

mnum the mapping register number

8.41.3.4 long iato::Rat::setmap (t_lreg *lreg*, const long *rnum*, const long *mnum*)

set the register mapping by type and id and return the old one

Parameters:

lreg the register type to set

rnum the register number to set

mnum the mapping register number

The documentation for this class was generated from the following file:

- Rat.hpp

8.42 iato::ReqBuf Class Reference

`#include <ReqBuf.hpp>`

Public Member Functions

- **ReqBuf** (void)
  create an empty buffer
- **ReqBuf** (**Mtx** ∗mtx)
- void **reset** (void)
  reset this buffer
- void **flush** (void)
  flush this buffer
- void **report** (void) const
  report this resource
- long **length** (void) const
  the buffer size
- PortReq ∗ **get** (const long index) const
  a port request by index
- void **add** (PortReq ∗preq)
- void **bind** (**Weakable** ∗pack)

8.42.1 Detailed Description

The **ReqBuf**(p. 281) class is the port request buffer class. It acts as an array of request ports. The request port buffer is used by the memory architecture to bind each request port with the equivalent port acknowledger in bypass mode.

8.42.2 Constructor & Destructor Documentation

8.42.2.1 iato::ReqBuf::ReqBuf (Mtx ∗ *mtx*)

create an empty buffer with a context

Parameters:
mtx the architectural context

8.42.3 Member Function Documentation

8.42.3.1 void iato::ReqBuf::add (PortReq ∗ *preq*)

add a port request to this buffer

Parameters:
preq the port request to add

8.42.3.2 void iato::ReqBuf::bind (Weakable ∗ *pack*)

bind a port acknowledger to all port request

Parameters:
pack the port acknowledger to bind

The documentation for this class was generated from the following file:

- ReqBuf.hpp

8.43 iato::Restart Class Reference

`#include <Restart.hpp>`

Public Member Functions

- **Restart** (void)
  create a new restart
- **Restart** (**Mtx** ∗mtx)
- void **reset** (void)
  reset this restart
- virtual void **flush** (void)=0
  flush all resource
- void **report** (void) const
  report this resource
- virtual void **bind** (**Runnable** ∗pipe, class Env ∗env)
- virtual bool **getsrlz** (void) const
  the serialization restart flag
- virtual bool **isvalid** (const t_octa iip, const long slot) const
- virtual bool **check** (const t_octa iip, const long slot)
- virtual void **pfdef** (void)
  flush everything by default
- virtual void **pflcl** (void)
  flush the pipe and restart with local conditions
- virtual void **pfstd** (const t_octa ip, const long slot)
- virtual void **pfsrl** (const t_octa ip, const long slot)
- virtual void **pfnxt** (const t_octa ip, const long slot)

Protected Attributes

- bool **d_ipfr**
  the instruction restart flag
- bool **d_srlz**
  the pre-serialization flag
- t_octa **d_rip**
  the restart ip
- long **d_slot**
  the restart slot

- **Runnable ∗ p_pipe**
  the pipeline to flush
- Register ∗ **p_rbk**
  the register bank

8.43.1 Detailed Description

The **Restart**(p. 283) class is a special class designed to act as a pipeline restart under certain circumstances. When a pipeline flush request occurs, all stages are flushed and a restart condition is placed in the restart. The fetch and decode stages uses that information to reset appropriately the associated resources and restart the execution of the instruction at the proper instruction slot.

8.43.2 Constructor & Destructor Documentation

8.43.2.1 iato::Restart::Restart (Mtx ∗ *mtx*)

create a new restart with a context

Parameters:
mtx the architectural context

8.43.3 Member Function Documentation

8.43.3.1 virtual void iato::Restart::bind (Runnable ∗ *pipe*, class Env ∗ *env*) [virtual]

bind the pipeline with this restart engine

Parameters:
pipe the pipeline to bind
env the resource environment

8.43.3.2 virtual bool iato::Restart::check (const t_octa *iip*, const long *slot*) [virtual]

check if an instruction must be cancelled and reset

Parameters:
iip the instruction ip to check
slot the slot to check

8.43.3.3 virtual bool iato::Restart::isvalid (const t_octa *iip*, const long *slot*) const [virtual]

check if an instruction is valid

Parameters:

iip the instruction ip to check

slot the slot to check

8.43.3.4 virtual void iato::Restart::pfnxt (const t_octa *ip*, const long *slot*) [virtual]

flush the pipe with a next restart condition

Parameters:

ip the offending ip

slot the offending slot

8.43.3.5 virtual void iato::Restart::pfsrl (const t_octa *ip*, const long *slot*) [virtual]

flush the pipe with a serialization condition

Parameters:

ip the offending ip

slot the offending slot

8.43.3.6 virtual void iato::Restart::pfstd (const t_octa *ip*, const long *slot*) [virtual]

flush the pipe with a standard restart condition

Parameters:

ip the offending ip

slot the offending slot

The documentation for this class was generated from the following file:

- Restart.hpp

8.44 iato::Rib Class Reference

`#include <Rib.hpp>`

Public Member Functions

- **Rib** (void)
  create a new rib
- **Rib** (**Mtx** ∗mtx)
- ∼**Rib** (void)
  destroy this rib
- void **reset** (void)
  reset this rib
- void **report** (void) const
  report this resource
- bool **isempty** (void) const
  true if the rib is empty
- bool **isvalid** (const long index) const
  true if the irb entry is valid
- void **clear** (const long index)
  clear an rib entry by idex
- bool **exists** (const Rid &rid) const
  true if an rid exists in this buffer
- void **add** (const Rid &rid)
- void **add** (const **Dsi** &inst)
- bool **isdep** (const **Dsi** &dsi) const
  if an instruction match a dependency

8.44.1 Detailed Description

The **Rib**(p. 286) class is the replay identification buffer. The class acts as a buffer that holds rid that can be use to mark instructions that needs to be replayed. The buffer holds a flags that permits to detect if an overflow occured in order to force a pipe flush.

8.44.2 Constructor & Destructor Documentation

8.44.2.1 iato::Rib::Rib (Mtx ∗ *mtx*)

create a new rib with a context

Parameters:

mtx the architectural context

8.44.3 Member Function Documentation

8.44.3.1 void iato::Rib::add (const Dsi & *inst*)

add an instruction dependency

Parameters:
inst the instruction to add

8.44.3.2 void iato::Rib::add (const Rid & *rid*)

add a new rid in the dependency list

Parameters:
rid the rid to add

The documentation for this class was generated from the following file:

- Rib.hpp

8.45 iato::Rob Class Reference

`#include <Rob.hpp>`

Public Member Functions

- **Rob** (void)
  create a new rob
- **Rob** (**Mtx** *mtx)
- ~**Rob** (void)
  destroy this rob
- void **reset** (void)
  reset this rob
- void **report** (void) const
  report this resource
- bool **isempty** (void) const
  true if the rob is empty
- bool **isfull** (void) const
  true if the rob is full
- bool **isvalid** (const long ridx) const
  true if the rob entry is valid
- bool **isvalid** (void) const
  true if the latest rob entry is valid
- bool **ispop** (void) const
  true if the latest rob entry is valid
- bool **issrlz** (void) const
  true if the latest rob entry is a serialization
- bool **isnop** (void) const
  true if the latest rob entry is a nop
- bool **isintr** (void) const
  true if the latest rob entry is an interrupt
- bool **iscancel** (void) const
  true if the latest rob entry has been cancelled
- void **alloc** (const long iiib)
- void **alloc** (const t_octa ip, const long slot, t_unit unit, const bool srlz, const bool nopb)

- long **alloc** (const **Dsi** &inst, const long imob, const long iiib)
- long **pop** (void)
  the latest irb and clean rob
- void **spop** (void)
  pop the latest serialize rob entry
- void **npop** (void)
  pop the latest nop rob entry
- long **ipop** (void)
  the latest interrupt index and clean rob
- t_octa **getiip** (void) const
  the latest instruction ip
- long **getslot** (void) const
  the latest instruction slot
- t_unit **getunit** (void) const
  the latest instruction unit
- long **getiidx** (void) const
  the latest irb index
- long **getimob** (void) const
  the latest memory ordering index
- long **getiiib** (void) const
  the latest iib index
- bool **getsbit** (void) const
  the latest speculative bit
- bool **getbbss** (void) const
  the latest branch speculative status
- void **setexe** (const long ridx, const bool flag)
- void **setcnlf** (const long ridx, const bool flag)
- void **setirb** (const long ridx, const long index)
- void **setintr** (const long ridx, const long iiib)
- void **setintr** (const long ridx, const long iiib, const bool sbit)
- void **setsbit** (const long ridx, const bool sbit)
- void **setbbss** (const long ridx, const bool bbss)

8.45.1 Detailed Description

The **Rob**(p. 288) class implements a reorder buffer as a fifo. The rob is used to retire instruction in order and process interrupt. This rob implementation is instruction based. Using an instruction based rob simplifies the architecture, since there is no need to handle MLX bundles as well as implied dependencies. The interrupt processing is also simplified. The entry allocation is on the other end complex.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 iato::Rob::Rob (Mtx * *mtx*)

create a new rob with a context

Parameters:

mtx the architectural context

8.45.3 Member Function Documentation

8.45.3.1 long iato::Rob::alloc (const Dsi & *inst*, const long *imob*, const long *iiib*)

allocate a new rob entry

Parameters:

inst the instruction

imob the memory ordering buffer index

iiib the instruction interrupt buffer index

8.45.3.2 void iato::Rob::alloc (const t_octa *ip*, const long *slot*, t_unit *unit*, const bool *srlz*, const bool *nopb*)

allocate a serialize rob entry with an ip and a slot

Parameters:

ip the ip to restart

slot the slot to restart

unit the instruction unit

srlz the serialize bit

nopb the nop bit

8.45.3.3 void iato::Rob::alloc (const long *iiib*)

allocate a new rob entry with an interrupt index

Parameters:

iiib the interrupt index

8.45.3.4 void iato::Rob::setbbss (const long *ridx*, const bool *bbss*)

set the bundle branch speculative status

Parameters:

ridx the rob index

bbss the speculative status

8.45.3.5 void iato::Rob::setcnlf (const long *ridx*, const bool *flag*)

set the cancel flag

Parameters:

ridx the rob index

flag the flag to set

8.45.3.6 void iato::Rob::setexe (const long *ridx*, const bool *flag*)

set the execute flag

Parameters:

ridx the rob index

flag the flag to set

8.45.3.7 void iato::Rob::setintr (const long *ridx*, const long *iiib*, const bool *sbit*)

set the interrupt entry with speculative info

Parameters:

ridx the rob index

iiib the virtual interrupt

sbit the speculative bit

8.45.3.8 void iato::Rob::setintr (const long *ridx*, const long *iiib*)

set the interrupt entry

Parameters:

ridx the rob index

iiib the interrupt index

8.45.3.9 void iato::Rob::setirb (const long *ridx*, const long *index*)

set the irb index entry

Parameters:

ridx the rob index

index the irb index

8.45.3.10 void iato::Rob::setsbit (const long *ridx*, const bool *sbit*)

set the speculative bit

Parameters:

ridx the rob index

sbit the speculative bit

The documentation for this class was generated from the following file:

- Rob.hpp

8.46 iato::RseStack Class Reference

`#include <RseStack.hpp>`

Public Member Functions

- **RseStack** (void)
  create a default rse stack
- **RseStack** (**Mtx** ∗mtx)
- ∼**RseStack** (void)
  destroy this rse stack
- void **reset** (void)
  reset this rse stack
- void **report** (void) const
  report this resource
- bool **isempty** (void) const
  true if the stack is empty
- bool **isfull** (void) const
  true if the stack is full
- void **push** (const Rse::State &state)
- Rse::State **pop** (void)
  the latest irb and clean rob
- void **shift** (void)
  shift the stack by one element

8.46.1 Detailed Description

The **RseStack**(p. 293) class implements a finite stack of rse state. When a rse dependant branch is made, the rse state is placed on the stack or retreived from it. The rse state contains mostly the cfm register that defines the state of the renaming circuitry. The rse stack is an essential piece of code that permits to operate with a branch predictor.

8.46.2 Constructor & Destructor Documentation

8.46.2.1 iato::RseStack::RseStack (Mtx ∗ *mtx*)

create a new rse stack with a context

Parameters:
mtx the architectural context

8.46.3 Member Function Documentation

8.46.3.1 void iato::RseStack::push (const Rse::State & *state*)

push a new rse state on the stack

Parameters:

state the state to push

The documentation for this class was generated from the following file:

- RseStack.hpp

8.47 iato::Runnable Class Reference

`#include <Runnable.hpp>`

Inherited by **iato::Delayable**, **iato::Hma**, **iato::Pipeline**, and **iato::System**.

Public Member Functions

- **Runnable** (void)
  create a default runnable
- **Runnable** (const string &name)
- virtual void **flush** (void)=0
  flush this runnable interface
- virtual void **run** (void)=0
  run this runnable object

Protected Attributes

- bool **d_flsh**
  the partial flush bit

8.47.1 Detailed Description

The **Runnable**(p. 295) class is an abstract class that defines the ”run” and ”flush” methods. The run method executes virtually one cycle of the runnbale object. The runnable interface can be reset and flushed. The difference is interface dependent.

8.47.2 Constructor & Destructor Documentation

8.47.2.1 iato::Runnable::Runnable (const string & *name*)

create a new runnable by name

Parameters:
name the runnable resource name

The documentation for this class was generated from the following file:

- Runnable.hpp

8.48 iato::Scoreboard Class Reference

`#include <Scoreboard.hpp>`

Public Types

- enum **t_model**

  the scoreboard model

Public Member Functions

- **Scoreboard** (void)

  create a new scoreboard

- **Scoreboard** (**Mtx** *mtx)
- **Scoreboard** (**Mtx** *mtx, const string &name)
- ~**Scoreboard** (void)

  destroy this scoreboard

- void **reset** (void)

  reset this scoreboard

- void **report** (void) const

  report this resource

- bool **ismarked** (const Rid &rid) const

  true if a register is marked

- void **mark** (const Rid &rid, const bool flag)
- void **setsr** (const **Ssi** &ssi)
- void **unsetsr** (const **Ssi** &ssi)
- void **lock** (const **Ssi** &ssi)
- void **unlock** (const **Ssi** &ssi)
- bool **ishazard** (const **Ssi** &ssi) const

  true if the instruction presents a hazard

8.48.1 Detailed Description

The **Scoreboard**(p. 296) class implements a register scoreboarding facility. By default, all Itanium ISa register can be monitored by the scoreboard with a simple lock or unlock mode. When a register instruction is marked in the scoreboard, the whole issue group might be put on hold until such register is unlocked. The locking model is by default the full model that provides scoreboarding for all register. Other models might be implemented lto support relazed model. The scoreboard also operates with an interrupted instruction index. The index is the oldest interrupted instruction in the instruction group.

8.48.2 Constructor & Destructor Documentation

8.48.2.1 iato::Scoreboard::Scoreboard (Mtx * *mtx*)

create a new scoreboard with a context

Parameters:

mtx the architectural context

8.48.2.2 iato::Scoreboard::Scoreboard (Mtx * *mtx*, const string & *name*)

create a new scoreboard with a context and a name

Parameters:

mtx the architectural context

name the resource name

8.48.3 Member Function Documentation

8.48.3.1 void iato::Scoreboard::lock (const Ssi & *ssi*)

lock all registers associated with an instruction

Parameters:

ssi the instruction to use for locking

8.48.3.2 void iato::Scoreboard::mark (const Rid & *rid*, const bool *flag*)

mark a register by rid and flag

Parameters:

rid the register id to set

flag the register flag to set

8.48.3.3 void iato::Scoreboard::setsr (const Ssi & *ssi*)

set a serialization condition

Parameters:

ssi the instruction to use for serialization

8.48.3.4 void iato::Scoreboard::unlock (const Ssi & *ssi*)

unlock all registers associated with an instruction

Parameters:

ssi the instruction to use for locking

8.48.3.5 void iato::Scoreboard::unsetsr (const Ssi & *ssi*)

unset a serialization condition

Parameters:

ssi the instruction to use for serialization

The documentation for this class was generated from the following file:

- Scoreboard.hpp

8.49 iato::Sct Class Reference

`#include <Sct.hpp>`

Public Types

- enum **t_sct**

  the two bits states

Public Member Functions

- **Sct** (void)

  create a default counter

- **Sct** (**t_sct** state)
- **Sct** (const **Sct** &that)
- void **reset** (void)

  reset this counter

- **Sct** & **operator=** (const **Sct** &that)
- bool **isstrong** (void) const

  true if the state is strong

- bool **isweak** (void) const

  true if the state is weak

- bool **istrue** (void) const

  true if the state is true

- void **update** (const bool flag)

8.49.1 Detailed Description

The **Sct**(p. 299) class is a two bit saturating counter used within prediction systems. The 'update' method can be used to adjust the counter value while the 'istrue' method returns true if the counter is in a true state.

8.49.2 Constructor & Destructor Documentation

8.49.2.1 iato::Sct::Sct (t_sct *state*)

create a counter with an initial state

Parameters:

state the initial state

8.49.2.2 iato::Sct::Sct (const Sct & *that*)

copy construct this counter

Parameters:
that the counter to copy

8.49.3 Member Function Documentation

8.49.3.1 Sct& iato::Sct::operator= (const Sct & *that*)

assign a counter to this one

Parameters:
that the counter to assign

8.49.3.2 void iato::Sct::update (const bool *flag*)

update a counter with a flag

Parameters:
flag the update flag

The documentation for this class was generated from the following file:

- Sct.hpp

8.50 iato::Slot Class Reference

`#include <Slot.hpp>`

Public Member Functions

- **Slot** (t_unit unit)
- **Slot** (t_unit unit, const long spos)
- **Slot** (const **Slot** &that)
- void **reset** (void)

  reset this slot

- **Slot** & **operator=** (const **Slot** &that)
- bool **isfree** (void) const

  true if the slot is free

- t_unit **getunit** (void) const

  the slot unit

- long **getspos** (void) const

  the slot position

- void **setspos** (const long spos)
- **Ssi getinst** (void) const

  the slot instruction

- **Ssi grabinst** (void)

  the slot instruction and clear the slot

- void **setinst** (const **Ssi** &ssi)

8.50.1 Detailed Description

the **Slot**(p. 301) class is an issue port slot. It is used to store a particular instruction beeing routed. The slot object holds the slot type as well as the slot index.

8.50.2 Constructor & Destructor Documentation

8.50.2.1 iato::Slot::Slot (t_unit *unit*)

create a slot by unit

Parameters:

unit the slot unit type

8.50.2.2 iato::Slot::Slot (t_unit *unit*, const long *spos*)

create a slot by unit and position

Parameters:
unit the slot unit type
spos the slot position

8.50.2.3 iato::Slot::Slot (const Slot & *that*)

copy construct this slot

Parameters:
that the slot to copy

8.50.3 Member Function Documentation

8.50.3.1 Slot& iato::Slot::operator= (const Slot & *that*)

assign a slot to this one

Parameters:
that the slot to assign

8.50.3.2 void iato::Slot::setinst (const Ssi & *ssi*)

set the slot instruction

Parameters:
ssi the instruction to set

8.50.3.3 void iato::Slot::setspos (const long *spos*)

set the slot position

Parameters:
spos the slot position to set

The documentation for this class was generated from the following file:

- Slot.hpp

8.51 iato::Spb Class Reference

`#include <Spb.hpp>`

Public Member Functions

- **Spb** (**Mtx** *mtx)
- **Spb** (**Mtx** *mtx, const string &name)
- ~**Spb** (void)

  destroy this buffer

- void **reset** (void)

  reset this buffer

- void **report** (void) const

  report this resource

- long **getsize** (t_unit unit) const

  a buffer size by unit

- void **add** (**Slot** *slot)

  add a slot in the buffer

- bool **isfree** (void) const

  true if all slots are free

- long **find** (const t_unit unit, const bool stb, const bool ldb) const
- void **setinst** (t_unit unit, const long slot, const **Ssi** &ssi)
- **Ssi getinst** (t_unit unit, const long slot) const

  an instruction by unit and slot

- bool **isintr** (void) const

  true if the port buffer has been interupted

- void **setintr** (const Interrupt &vi, const long iioi)

Protected Attributes

- long **d_mbsz**

  the M buffer size

- long **d_ldsn**

  the number of load slots

- long **d_stsn**

  the number of store slots

- **Slot** ** **p_mbuf**

  the M slot buffer

- long **d_ibsz**
  the I buffer size
- **Slot** ∗∗ **p_ibuf**
  the I slot buffer
- long **d_fbsz**
  the F buffer size
- **Slot** ∗∗ **p_fbuf**
  the F slot buffer
- long **d_bbsz**
  the B buffer size
- **Slot** ∗∗ **p_bbuf**
  the B slot buffer
- Interrupt **d_intr**
  the global interrupt
- long **d_iioi**
  the instruction index

8.51.1 Detailed Description

The **Spb**(p. 303) class is the slot port buffer. It is a set of ports that are grouped into a family set. At construction, the context determines how many slots can be held in the buffer. Each micro-pipeline is responsible to add the required slot. The **Spb**(p. 303) is used by an in-order simulation engine an issue buffer or as an output buffer after the renaming stage. The port buffer also provides a special slot for global interrupt within a group. The interrupt and the offending instruction order index is stored as well.

8.51.2 Constructor & Destructor Documentation

8.51.2.1 iato::Spb::Spb (Mtx ∗ *mtx*)

create a new buffer with a context

Parameters:
mtx the architectural context

8.51.2.2 iato::Spb::Spb (Mtx ∗ *mtx*, const string & *name*)

create a new buffer with a context and a name

Parameters:
mtx the architectural context
name the buffer name

8.51.3 Member Function Documentation

8.51.3.1 long iato::Spb::find (const t_unit *unit*, const bool *stb*, const bool *ldb*) const

find a free slot by unit and store/load bits

Parameters:

unit the unit slot to find

stb the store bit

ldb the load bit

8.51.3.2 void iato::Spb::setinst (t_unit *unit*, const long *slot*, const Ssi & *ssi*)

set the slot instruction by unit, index and instruction

Parameters:

unit the unit slot to use

slot the slot index

ssi the instruction to set

8.51.3.3 void iato::Spb::setintr (const Interrupt & *vi*, const long *iioi*)

set the buffer interrupt and index

Parameters:

vi the offending interrupt

iioi the offending index

The documentation for this class was generated from the following file:

- Spb.hpp

8.52 iato::Ssi Class Reference

`#include <Ssi.hpp>`

Inherited by **iato::Dsi**.

Public Member Functions

- **Ssi** (void)
  create a default ssi
- **Ssi** (const Instr &inst)
- **Ssi** (const **Ssi** &that)
- **Ssi** & **operator=** (const **Ssi** &that)
- **Ssi** & **operator=** (const Instr &that)
- void **reset** (void)
  reset this ssi
- bool **ispresr** (void) const
  true if this instruction needs pre-serialization
- bool **ispostsr** (void) const
  true if this instruction needs post-serialization
- void **setrix** (const long index)
- long **getrix** (void) const
  the reorder index
- void **setiib** (const long index)
- long **getiib** (void) const
  the iib index
- void **setcnlf** (const bool cnlf)
- void **setcnlf** (const Rid &rid, const bool cnlf)
- bool **getcnlf** (void) const
  the cancellation flag
- void **setintr** (const bool intr)
- bool **getintr** (void) const
  the interrupt flag
- void **setmofl** (const bool mofl)
- bool **getmofl** (void) const
  the memory ordering flag
- void **setvspf** (const bool vspf)
- bool **getvspf** (void) const
  the valid speculation flag
- void **setppvl** (const bool ppvl)

- bool **getppvl** (void) const

  the predicted predicate value

- void **setppfl** (const bool ppfl)
- bool **getppfl** (void) const

  the predicate prediction flag

- void **setxflg** (const bool xflg)
- bool **getxflg** (void) const

  the stat extra flag

- void **setiste** (const Rse::State &state)
- Rse::State **getiste** (void) const

  the instruction rse state

- Cfm **geticfm** (void) const

  the instruction cfm

- void **setsste** (const Rse::State &state)
- Rse::State **getsste** (void) const

  the speculative rse state

- Cfm **getscfm** (void) const

  the speculative cfm

Protected Attributes

- long **d_ridx**

  the reorder index

- long **d_iidx**

  the iib index

- bool **d_cnlf**

  the cancellation flag

- bool **d_intr**

  the interrupt flag

- bool **d_mofl**

  the memory ordering flag

- bool **d_vspf**

  the valid speculation flag

- bool **d_ppvl**

  the predicted predicate value

- bool **d_ppfl**

  the predicate prediction flag

- bool **d_xflg**

  the extra stat flag

- Rse::State **d_iste**

  the instruction rse state

- Rse::State **d_sste**

  the speculative rse state

8.52.1 Detailed Description

The **Ssi**(p. 306) class is the sta=tically scheduled instruction that is derived from the isa instruction class. The class holds additional instruction information that are used in a speculative environment. One particular important information is the speculative cfm value that is vaed in the instruction.

8.52.2 Constructor & Destructor Documentation

8.52.2.1 iato::Ssi::Ssi (const Instr & *inst*)

create a ssi from an instruction

Parameters:
inst the instruction to use

8.52.2.2 iato::Ssi::Ssi (const Ssi & *that*)

copy construct this ssi

Parameters:
that the ssi to copy

8.52.3 Member Function Documentation

8.52.3.1 Ssi& iato::Ssi::operator= (const Instr & *that*)

assign an instruction to this ssi

Parameters:
that the instruction to assign

Reimplemented in **iato::Dsi** (p. 218).

8.52.3.2 Ssi& iato::Ssi::operator= (const Ssi & *that*)

assign a ssi to this one

Parameters:
that the ssi to assign

8.52.3.3 void iato::Ssi::setcnlf (const Rid & *rid*, const bool *cnlf*)

set the instruction cancellation flag by rid

Parameters:
rid the rid to use for cancellation
cnlf the cancel flag to set

8.52.3.4 void iato::Ssi::setcnlf (const bool *cnlf*)

set the instruction cancellation flag

Parameters:
cnlf the cancel flag to set

8.52.3.5 void iato::Ssi::setiib (const long *index*)

set the iib index

Parameters:
index the iib index

8.52.3.6 void iato::Ssi::setintr (const bool *intr*)

set the interrupt flag flag

Parameters:
intr the interrupt flag to set

8.52.3.7 void iato::Ssi::setiste (const Rse::State & *state*)

set the instruction rse state

Parameters:
state the state to set

8.52.3.8 void iato::Ssi::setmofl (const bool *mofl*)

set the memory ordering flag

Parameters:
mofl the ordering flag to set

8.52.3.9 void iato::Ssi::setppfl (const bool *ppfl*)

set the predicate prediction flag

Parameters:
ppfl the prediction flag to set

8.52.3.10 void iato::Ssi::setppvl (const bool *ppvl*)

set the predicated predicate value

Parameters:
ppvl the predicted value to set

8.52.3.11 void iato::Ssi::setrix (const long *index*)

set the reorder index

Parameters:
index the reorder index

8.52.3.12 void iato::Ssi::setsste (const Rse::State & *state*)

set the speculative rse state

Parameters:
state the state to set

8.52.3.13 void iato::Ssi::setvspf (const bool *vspf*)

set the valid speculation flag

Parameters:
vspf the speculation flag to set

8.52.3.14 void iato::Ssi::setxflg (const bool *xflg*)

set the stat extra flag

Parameters:
xflg the extra flag to set

The documentation for this class was generated from the following file:

- Ssi.hpp

8.53 iato::Stage Class Reference

`#include <Stage.hpp>`

Inherits **iato::Delayable**.

Inherited by **iato::Pipelane**.

Public Member Functions

- **Stage** (**Mtx** ∗mtx, const string &name)
- **Stage** (**Mtx** ∗mtx, const long sidx, const string &name)
- void **reset** (void)

  reset this stage

- void **flush** (void)

  flush this stage

- void **run** (void)

  run this stage

- virtual bool **ishalted** (void) const

  return true if the stage is halted

- virtual bool **isholding** (void) const

  the holding bit status

- virtual void **setprev** (Env ∗env, **Stage** ∗stg)
- virtual void **setnext** (Env ∗env, **Stage** ∗stg)
- virtual void **bind** (Env ∗env, **Stage** ∗pstg, **Stage** ∗nstg)

Protected Attributes

- bool **d_halt**

  the halt bit

- long **d_sidx**

  the stage index

- **Stage** ∗ **p_pstg**

  the previous stage

- **Stage** ∗ **p_nstg**

  the next stage

8.53.1 Detailed Description

The **Stage**(p. 311) class is an abstract class that models a pipeline stage. At construction a stage uses a context object to configure itself. Once all stages have been created, they can be bounded together with the 'bind' method. The bind method uses an environment object which among other things holds the global resources of the processor. A stage is defined with a name. A hold bit also indicates if a stage has been put on hold by another stage. **Pipeline**(p. 268) stages are linked together in the mean of the previous and next stage.

8.53.2 Constructor & Destructor Documentation

8.53.2.1 iato::Stage::Stage (Mtx * *mtx*, const string & *name*)

create a new stage by context and name

Parameters:

mtx the architectural context

name the stage name

8.53.2.2 iato::Stage::Stage (Mtx * *mtx*, const long *sidx*, const string & *name*)

create a new stage by context, index and name

Parameters:

mtx the architectural context

sidx the stage index

name the stage name

8.53.3 Member Function Documentation

8.53.3.1 virtual void iato::Stage::bind (Env * *env*, Stage * *pstg*, Stage * *nstg*) [virtual]

bind this stage with an execution environment

Parameters:

env the execution environment

pstg the previous stage

nstg the next stage

Reimplemented in **iato::Pipelane** (p. 266).

8.53.3.2 virtual void iato::Stage::setnext (Env * *env*, Stage * *stg*) [virtual]

set the next stage

Parameters:

env the execution environment

stg the previous stage to set

8.53.3.3 virtual void iato::Stage::setprev (Env * *env*, Stage * *stg*) [virtual]

set the previous stage

Parameters:

env the execution environment

stg the previous stage to set

The documentation for this class was generated from the following file:

- Stage.hpp

8.54 iato::Station Class Reference

`#include <Station.hpp>`

Public Member Functions

- **Station** (const t_unit unit)
- **Station** (**Mtx** *mtx, const t_unit unit)
- **Station** (**Mtx** *mtx, const t_unit unit, const string &name)
- ~**Station** (void)
  destroy this station table
- void **reset** (void)
  reset the station table
- void **report** (void) const
  report this resource
- void **setgcs** (const long index)
- long **getgcs** (void) const
  the station gcs index
- bool **isfull** (void) const
  true if the table is full
- bool **isempty** (void) const
  true if the table is empty
- long **alloc** (const **Dsi** &dsi)
- void **clear** (const long index)
- void **setrdy** (const Rid &rid)
- **Dsi getrdy** (void)
  the next instruction ready for execution
- void **setcnl** (const Rid &rid, const bool value)
- void **resched** (const long index)
- void **setpnrd** (const long index, const bool pnrd)
- void **dump** (void) const
  dump the station content

8.54.1 Detailed Description

The **Station**(p. 314) class is a resource that is designed to act as a reservation station table. A reservation station is one entry in the table. The sole purpose of a resrvationn station is to park instruction waiting for their operands. When the operands are ready, the instruction is waked-up and delivered to the next stage. The complete administration task is actually quite complexe. First, a valid bit indicates if the entry is valid. Second a wake-up bit indicates if the instruction has been waked-up. The instruction remains in the station until it has been write-backed or cancelled by local or global flush. The station also maintains a priority table that is used by the scheduler to select the ready instructions.

8.54.2 Constructor & Destructor Documentation

8.54.2.1 iato::Station::Station (const t_unit *unit*)

create a station table by type

Parameters:
unit the station unit type

8.54.2.2 iato::Station::Station (Mtx * *mtx*, const t_unit *unit*)

create a station table by type and context

Parameters:
mtx the architectural context
unit the station unit type

8.54.2.3 iato::Station::Station (Mtx * *mtx*, const t_unit *unit*, const string & *name*)

create a station table by type and context and name

Parameters:
mtx the architectural context
unit the station unit type
name the station name

8.54.3 Member Function Documentation

8.54.3.1 long iato::Station::alloc (const Dsi & *dsi*)

alloc a new entry in the table by instruction

Parameters:
dsi the instruction to insert

8.54.3.2 void iato::Station::clear (const long *index*)

clear a station entry by index

Parameters:
index the station index

8.54.3.3 void iato::Station::resched (const long *index*)

reschedule an instruction by index

Parameters:
index the instruction station index

8.54.3.4 void iato::Station::setcnl (const Rid & *rid*, const bool *value*)

mark instruction cancelled by rid and value

Parameters:

rid the rid to check

value the cancel flag

8.54.3.5 void iato::Station::setgcs (const long *index*)

set the station gcs index

Parameters:

index the index to set

8.54.3.6 void iato::Station::setpnrd (const long *index*, const bool *pnrd*)

mark the predicate not ready flag

Parameters:

index the instruction station index

pnrd the predicate not ready flag

8.54.3.7 void iato::Station::setrdy (const Rid & *rid*)

set the station ready bit by rid

Parameters:

rid the rid to compare

The documentation for this class was generated from the following file:

- Station.hpp

8.55 iato::Stb Class Reference

`#include <Stb.hpp>`

Public Member Functions

- **Stb** (void)
  create a new stb
- **Stb** (**Mtx** *mtx)
- ~**Stb** (void)
  destroy this stb
- void **reset** (void)
  reset this stb
- void **report** (void) const
  report this resource
- void **bind** (Memory *mem)
- bool **isempty** (void) const
  true if the stb is empty
- void **push** (t_mreq type, const t_octa addr, const t_octa data)
- void **push** (t_mreq type, const t_octa addr, const t_real &data)

8.55.1 Detailed Description

The **Stb**(p. 317) class is the store buffer. The store buffer is a queue of pair (address/value) and a size. Entries are added in order in the buffer. The store buffer is primarily used at the commit stage to perform the write operations.

8.55.2 Constructor & Destructor Documentation

8.55.2.1 iato::Stb::Stb (Mtx * *mtx*)

create a new stb with a context

Parameters:
mtx the architectural context

8.55.3 Member Function Documentation

8.55.3.1 void iato::Stb::bind (Memory * *mem*)

bind a bypass memory to this buffer

Parameters:
mem the bypass memory to bind

8.55.3.2 void iato::Stb::push (t_mreq *type*, const t_octa *addr*, const t_real & *data*)

push a store request in the store buffer

Parameters:

type the request type

addr the request address

data the request data

8.55.3.3 void iato::Stb::push (t_mreq *type*, const t_octa *addr*, const t_octa *data*)

push a store request in the store buffer

Parameters:

type the request type

addr the request address

data the request data

The documentation for this class was generated from the following file:

- Stb.hpp

8.56 iato::System Class Reference

`#include <System.hpp>`

Inherits **iato::Runnable**.

Public Member Functions

- **System** (**Mtx** *mtx, const string &name)
- **System** (**Mtx** *mtx, const string &name, const vector< string > &argv)
- ~**System** (void)
  destroy this system interface
- void **reset** (void)
  reset this system interface
- void **flush** (void)
  flsuh this system interface
- void **run** (void)
  run this system interface
- void **report** (void) const
  report some system information
- Syscall * **getsci** (void) const
  the system call plugin
- **Hma** * **gethma** (void) const
  the memory model architecture
- t_octa **getentry** (void) const
  the entry point
- t_octa **getstkva** (void) const
  the top of stack
- t_octa **getbspva** (void) const
  the backing store base
- Checker * **getchecker** (void) const
  the elf image checker

8.56.1 Detailed Description

The **System**(p. 319) class is the memory system interface. The class is used to bind the system memory as well as other interface that might needed during the course of the simulation. With a program name, a program image is created and associated with various memory unit. The binding between an instruction memory and a program image is done during the simulation binding process.

8.56.2 Constructor & Destructor Documentation

8.56.2.1 iato::System::System (Mtx * *mtx*, const string & *name*)

create a new system interface by context and program name

Parameters:

mtx the architectural context

name the program name

8.56.2.2 iato::System::System (Mtx * *mtx*, const string & *name*, const vector< string > & *argv*)

create a new system interface by context, program name and arguments

Parameters:

mtx the architectural context

name the program name

argv the program arguments

The documentation for this class was generated from the following file:

- System.hpp

8.57 iato::Trb Class Reference

`#include <Trb.hpp>`

Public Member Functions

- **Trb** (void)

  create a default translation bank

- **Trb** (**Mtx** *mtx)
- **Trb** (**Mtx** *mtx, const string &name)
- ~**Trb** (void)

  destroy this bank

- void **reset** (void)

  reset this bank

- void **report** (void) const

  report this resource

- bool **isready** (const long vnum) const

  true if a trb entry is ready

- bool **gettbit** (const long vnum) const

  true if a translation bit is set

- void **settnum** (const long vnum, const long tnum, const bool rbit)
- long **gettnum** (const long vnum) const

  the translation number

- long **initial** (void)

  allocate an initial translation

- long **alloc** (void)

  the index of a new allocated trb entry

- long **clean** (const long vnum, const long onum)
- void **cancel** (const long vnum, const long onum)
- void **setrdy** (const long vnum)

8.57.1 Detailed Description

The **Trb**(p. 321) class is the translation register bank class definition. For each register type a translation bank is created with a size equal to the number of physical registers. Each register provides a direct translation to the physical register. A valid bit, associated with the register indicates if that translation register is valid. At reset, the default translation is initialized with a mapping corresponding to the logical register definition.

8.57.2 Constructor & Destructor Documentation

8.57.2.1 iato::Trb::Trb (Mtx * *mtx*)

create a translation bank by context

Parameters:
mtx the architectural context

8.57.2.2 iato::Trb::Trb (Mtx * *mtx*, const string & *name*)

create a translation bank by context and name

Parameters:
mtx the architectural context
name the rat resource name

8.57.3 Member Function Documentation

8.57.3.1 void iato::Trb::cancel (const long *vnum*, const long *onum*)

cancel a trb translation by forcing the ready bit

Parameters:
vnum the virtual number
onum the old virtual number

8.57.3.2 long iato::Trb::clean (const long *vnum*, const long *onum*)

clean a trb entry by number and return the old translation

Parameters:
vnum the virtual number
onum the old virtual number

8.57.3.3 void iato::Trb::setrdy (const long *vnum*)

force the ready bit by index

Parameters:
vnum the virtual number

8.57.3.4 void iato::Trb::settnum (const long *vnum*, const long *tnum*, const bool *rbit*)

set the trb translation number

Parameters:
vnum the virtual number

tnum the translation number

rbit the ready bit

The documentation for this class was generated from the following file:

- Trb.hpp

8.58 iato::Urb Class Reference

`#include <Urb.hpp>`

Public Member Functions

- **Urb** (void)

  create a default universal bank

- **Urb** (**Mtx** ∗mtx)
- **Urb** (**Mtx** ∗mtx, const string &name)
- ∼**Urb** (void)

  destroy this bank

- void **reset** (void)

  reset this bank

- void **report** (void) const

  report this resource

- void **setuvr** (const long rnum, const Uvr &uvr)
- Uvr **getuvr** (const long rnum) const

  the uvr value by index

- long **alloc** (void)

  the index of a new allocated urb entry

- void **clean** (const long rnum)

8.58.1 Detailed Description

The **Urb**(p. 324) class is the universal register bank class definition. An urb entry holds a uvr value and a valid bit. The urb is used to store in-flight instruction register result before the value is commited into the logical register file. The size of the urb is an architectural parameter.

8.58.2 Constructor & Destructor Documentation

8.58.2.1 iato::Urb::Urb (Mtx ∗ *mtx*)

create a universal bank by context

Parameters:

mtx the architectural context

8.58.2.2 iato::Urb::Urb (Mtx * *mtx*, const string & *name*)

create a universal bank by context and name

Parameters:

mtx the architectural context

name the rat resource name

8.58.3 Member Function Documentation

8.58.3.1 void iato::Urb::clean (const long *rnum*)

clean an urb entry param rnum the register to clean

8.58.3.2 void iato::Urb::setuvr (const long *rnum*, const Uvr & *uvr*)

set the urb entry by index and value

Parameters:

rnum the register number

uvr the uvr value to set

The documentation for this class was generated from the following file:

- Urb.hpp

8.59 iato::Urf Class Reference

`#include <Urf.hpp>`

Public Member Functions

- **Urf** (void)

  create a default urf

- **Urf** (**Mtx** ∗mtx)
- **Urf** (**Mtx** ∗mtx, const string &name)
- ∼**Urf** (void)

  destroy this urf

- void **reset** (void)

  reset this urf

- void **report** (void) const

  report this resource

- void **flush** (void)

  flush this urf

- **Rat** ∗ **getrat** (void) const

  the associated rat

- **Trb** ∗ **gettrb** (void) const

  the associated trb

- **Urb** ∗ **geturb** (void) const

  the associated trb

- bool **isready** (const Rid &rid) const

  true if a trb entry is ready

- bool **isready** (const Operand &oprd) const

  true if an operand is ready

- Uvr **eval** (const Rid &rid) const
- void **eval** (Operand &oprd) const
- void **reroute** (const Rid &rid)
- void **reroute** (const Result &resl)
- void **update** (const Result &resl)
- void **clean** (const Result &resl)
- void **cancel** (const Result &resl)

8.59.1 Detailed Description

The **Urf**(p. 326) class is the universal register file class definition. The **Urf**(p. 326) is built with a translation register bank (trb), the universal register bank (urb), and the register alias table (rat). The main reason to group these three resources into one, is to simplify the operand evalution and the result update as well as the speculative flush of the renaming logic. Each individual resources can be accessed by getting them.

8.59.2 Constructor & Destructor Documentation

8.59.2.1 iato::Urf::Urf (Mtx * *mtx*)

create a urf by context

Parameters:

mtx the architectural context

8.59.2.2 iato::Urf::Urf (Mtx * *mtx*, const string & *name*)

create a urf by context and name

Parameters:

mtx the architectural context

name the rat resource name

8.59.3 Member Function Documentation

8.59.3.1 void iato::Urf::cancel (const Result & *resl*)

cancel the urf with a result

Parameters:

resl the result used for update

8.59.3.2 void iato::Urf::clean (const Result & *resl*)

clean the urf with a result

Parameters:

resl the result used for cleanup

8.59.3.3 void iato::Urf::eval (Operand & *oprd*) const

evaluate an operand in the urf

Parameters:

oprd the operand to evaluate

8.59.3.4 Uvr iato::Urf::eval (const Rid & *rid*) const

evaluate a rid in the urf

Parameters:
rid the rid to evaluate

8.59.3.5 void iato::Urf::reroute (const Result & *resl*)

reroute all result registers

Parameters:
resl the result used for rerouting

8.59.3.6 void iato::Urf::reroute (const Rid & *rid*)

reroute a rid in the urf

Parameters:
rid the rid to reroute

8.59.3.7 void iato::Urf::update (const Result & *resl*)

update the urf with a result

Parameters:
resl the result used for update

The documentation for this class was generated from the following file:

- Urf.hpp

8.60 iato::Watchdog Class Reference

`#include <Watchdog.hpp>`

Public Member Functions

- **Watchdog** (void)
  create a new watchdog
- **Watchdog** (**Mtx** ∗mtx)
- void **reset** (void)
  reset this watchdog
- void **report** (void) const
  report this resource
- void **notify** (void)
  notify this watchdog

8.60.1 Detailed Description

The **Watchdog**(p. 329) class is a simple class used to detect infinite loop. The watch dog operate with a counter which is decreased at each cycle. When a particular stage judge pertinent to say that it is alive it notify the watchdog which reload its counter. The watchdog is initialized with a 100 default cycle counter.

8.60.2 Constructor & Destructor Documentation

8.60.2.1 iato::Watchdog::Watchdog (Mtx ∗ *mtx*)

create a new watchdog with a context

Parameters:
mtx the architectural context

The documentation for this class was generated from the following file:

- Watchdog.hpp

8.61 iato::Weakable Class Reference

`#include <Weakable.hpp>`

Inherits **iato::Delayable**.

Public Member Functions

- **Weakable** (void)
  create a default weakable
- **Weakable** (const string &name)
- virtual void **notify** (Resource *res)=0

8.61.1 Detailed Description

The **Weakable**(p. 330) class is an abstract class that defines the ”notify” method. The ’notify’ method is used by an object to request an immediate action. Most likely, the notify method will invoke the activate method, although the notify method has an argument that is the caller.

8.61.2 Constructor & Destructor Documentation

8.61.2.1 iato::Weakable::Weakable (const string & *name*)

create a new weakable by name

Parameters:
name the weakable resource name

8.61.3 Member Function Documentation

8.61.3.1 virtual void iato::Weakable::notify (Resource * *res*) [pure virtual]

notify this weakable interface

Parameters:
res the resource notifier

The documentation for this class was generated from the following file:

- Weakable.hpp

Part V

ECU Library

Chapter 9

ECU Library Compound Index

9.1 IATO ECU LIBRARY Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 10

ECU Library Reference

10.1 iato::Fetcher Class Reference

`#include <Fetcher.hpp>`

Public Member Functions

- **Fetcher** (void)
  create a default fetcher
- **Fetcher** (**Utx** ∗utx)
- ∼**Fetcher** (void)
  destroy this fetcher
- void **reset** (void)
  reset this fetcher
- void **bind** (Memory ∗imem)
- bool **isempty** (void) const
  true if no bundle is available
- void **fill** (const t_octa ip)
- void **refill** (const t_octa ip, const long slot)
- void **pack** (void)
  pack the tech buffer
- Bundle **getbndl** (void)
  the next available bundle

10.1.1 Detailed Description

The **Fetcher**(p. 335) class is a base class used to accumulate bundles prior their dispersion. The number of bundles that can be processed depends on the issue width in disperse mode or is one by default. The class is designed to operate within an emulation environment.

10.1.2 Constructor & Destructor Documentation

10.1.2.1 iato::Fetcher::Fetcher (Utx * *utx*)

create a fetcher by context

Parameters:
utx the utility context

10.1.3 Member Function Documentation

10.1.3.1 void iato::Fetcher::bind (Memory * *imem*)

bind the instruction memory

Parameters:
imem the instruction memory to bind

10.1.3.2 void iato::Fetcher::fill (const t_octa *ip*)

fill the fetch buffer at a certain ip

Parameters:
ip the ip used for fetching

10.1.3.3 void iato::Fetcher::refill (const t_octa *ip*, const long *slot*)

refill the fetch buffer at a certain ip and slot

Parameters:
ip the ip used for fetching
slot the restarting slot

The documentation for this class was generated from the following file:

- Fetcher.hpp

10.2 iato::Mapper Class Reference

`#include <Mapper.hpp>`

Public Types

- enum **t_tmem**

  the mapper memory types

Public Member Functions

- **Mapper** (void)

  create a default mapper

- **Mapper** (Mtx *mtx)
- **Mapper** (Mtx *mtx, const string &name)
- void **reset** (void)

  reset this mapper

- void **bind** (Memory *mem)
- void **bind** (**t_tmem** type, Memory *mem)
- void **process** (Mrt &mrt)
- void **update** (Result &result)

10.2.1 Detailed Description

The **Mapper**(p. 337) class is a memory mapper class that adapts a memory transaction with a result object. Additionally, the adapter is designed respond to alat request.

10.2.2 Constructor & Destructor Documentation

10.2.2.1 iato::Mapper::Mapper (Mtx * *mtx*)

create a mapper with a context

Parameters:

mtx the architectural context

10.2.2.2 iato::Mapper::Mapper (Mtx * *mtx*, const string & *name*)

create a mapper with a context and a name

Parameters:

mtx the architectural context

name the resource name

10.2.3 Member Function Documentation

10.2.3.1 void iato::Mapper::bind (t_tmem *type*, Memory * *mem*)

bind a memory to the mapper by type

Parameters:

type the memory type to bind

mem the memory to bind

10.2.3.2 void iato::Mapper::bind (Memory * *mem*)

bind a memory to the mapper

Parameters:

mem the memory to bind

10.2.3.3 void iato::Mapper::process (Mrt & *mrt*)

process a memory request

Parameters:

mrt the memory request to process

10.2.3.4 void iato::Mapper::update (Result & *result*)

update a result with a memory request

Parameters:

result the result to update

The documentation for this class was generated from the following file:

- Mapper.hpp

10.3 iato::Renamer Class Reference

`#include <Renamer.hpp>`

Public Member Functions

- **Renamer** (void)

  create a default renamer

- **Renamer** (**Utx** ∗utx)
- void **bind** (Memory ∗mem, Register ∗rbk)
- void **preset** (const Instr &inst)
- void **aftset** (const Instr &inst)
- void **update** (const Result &resl)

10.3.1 Detailed Description

The **Renamer**(p. 339) class is the rse implementation with direct memory access. The class can be built with a context and the memory is bind to the object at run time. The register bank is also used here to access the bsp and bspstore values. The renamer class is designed to operate with an emulation environment.

10.3.2 Constructor & Destructor Documentation

10.3.2.1 iato::Renamer::Renamer (Utx ∗ *utx*)

create a renamer with a context

Parameters:

utx the utility context

10.3.3 Member Function Documentation

10.3.3.1 void iato::Renamer::aftset (const Instr & *inst*)

after set the rse state with an instruction

Parameters:

inst the instruction used to preset

10.3.3.2 void iato::Renamer::bind (Memory ∗ *mem*, Register ∗ *rbk*)

bind the memory and the register bank to this rse

Parameters:

mem the memory to bind

rbk the register bank

10.3.3.3 void iato::Renamer::preset (const Instr & *inst*)

preset the rse state with an instruction

Parameters:
inst the instruction used to preset

10.3.3.4 void iato::Renamer::update (const Result & *resl*)

update the rse state with a result

Parameters:
resl the result used to update

The documentation for this class was generated from the following file:

- Renamer.hpp

10.4 iato::Utx Class Reference

`#include <Utx.hpp>`

Public Member Functions

- **Utx** (void)

  create a new context

- void **reset** (void)

  reset this context

- void **update** (const t_arch arch)

10.4.1 Detailed Description

The **Utx**(p. 341) class is the utility extension context class. This class is derived from the Mtx context class and provides additional parameters that are used by the extension implementation.

10.4.2 Member Function Documentation

10.4.2.1 void iato::Utx::update (const t_arch *arch*)

update this contect with a particular archietcture

Parameters:

arch the archietcture used for update

The documentation for this class was generated from the following file:

- Utx.hpp

Index