Contents

HSA PRM Conformance Manual

HSA PRM Specification revision: 1.0 Provisional (2014-06-05)

Test suite revision: 2014-12-12 (CL 1104888)

Overview

HSA PRM Conformance test suite is used to validate an implementation of Heterogeneous System Architecture Intermediate Language (HSAIL) virtual machine and language as described in “HSA Programmer’s Reference Manual”. The focus is on verification of ISA produced by HSAIL Finalizer and functionality of kernels on the agent. HSAIL Linking extension is used to finalize test programs while HSA Core Runtime is used to set up and invoke test kernels and to provide certain functionality for test code on the host (for some tests).

A test consists of scenario which describes test data, HSAIL program(s) in BRIG format, code, dispatches and validation. The test is deemed passed if all scenario steps (including validation) are successful. Many basic tests for HSAIL instruction contain just one program with one kernel, one or several input and output buffers, one dispatch and validation of output buffer. An example of such test is the test for `add_u32` HSAIL instruction with register operands: one kernel, two input buffers and one output buffer.

Note that each test actually expands into several testcases, one for each valid combination of parameters. These parameters are described in Test details section. For example, the test for abs instruction consists of several tests for `f32` and `f64` types.

Building test suite

HSA PRM Conformance test suite uses CMake. The following dependencies need to be made available before the build:

- `HSAIL-Tools` can be obtained from HSA github repository. `HSAIL-Tools-PATH` needs to be set to point to it.
- HSA Runtime and Finalizer extension includes. These can be obtained from HSA github repository as well. `HSA-Runtime-Inc-PATH` needs to be set to point to directory with `hsa.h`. `HSA-Runtime-Ext-PATH` needs to be set to point to directory with `hsa_ext_finalize.h`.

The test suite is known to build on Windows and Linux. cmake_windows.sh and cmake_linux.sh scripts contain examples of command lines that can be used to run CMake on Windows and Linux.

Running test suite

The test suite can be invoked as "hcprm" program that runs on the host.

To run the suite, make sure to have a dynamic HSA RT libraries with corresponding bitness in the search path (e.g. `PATH` variable for Windows, `LD_LIBRARY_PATH` - for Linux).

A simple scenario of running all the tests on Linux 64-bit may look like:

```
export LD_LIBRARY_PATH="$HSA_RT_LIB:$LD_LIBRARY_PATH"
bin/lnx64a/hcprm
  -tests prm/
  -exclude amdhsa.exclude
  -runner hrunner
  -verbose
  -testlog results_linux64.log
```

where `$HSA_RT_LIB` points to the HSA RT libraries, e.g. `/compiler/dist/Obsidian/dist/linux/debug.hsa_foundation/lib/x86_64`.

For convenience the package also contains Linux and Windows batch files, which take path to the HSA Runtime binaries and run the whole suite.

The following example demonstrates how to run these scripts on Linux. The results grouped by sub-suites will be displayed during the run along with overall statistics, per test results can be found in results_linux64.log:

```
$ ./run.sh /compiler/dist/Obsidian/dist/linux/debug.hsa_foundation/
```

```
prm/arithmetic/intfp/abs
  Passed:    56   Failed:    0   Error:    0   NA:    0   Total:    56
prm/arithmetic/intfp/add
  Passed:   255   Failed:    0   Error:    0   NA:    0   Total:   255
...
prm/special/misc/laneid
  Passed:    24   Failed:    0   Error:    0   NA:    0   Total:    24

Testrun
  Passed: 13729   Failed:    0   Error:    0   NA:    0   Total: 13729
```

Using `-test` it is possible to specify a smaller set of tests, while `-exclude` allows to exclude tests (e.g. failing due to issues with the current HSA RT implementation).

Command line options

"hcprm" program supports the following command line options:

- `-tests TestSet`: prefix of tests to run, e.g. `-tests prm/` to run all the tests;
- `-exclude File`: file containing a list of test prefixes to be excluded from testing;
- `-verbose`: enables detailed test output in a log file;
- `-testlog File`: name for a log file, the default name is test.log;
- `-runner hrunner`: a mode of test grouping. By default, tests are not grouped. See also option `-testloglevel` which also affects grouping.
- `-testloglevel`: test grouping depth, the default is 4. See also `-runner` option;
- `-dump`: dump HSAIL and BRIG test sources for each test under corresponding folder (prm/...);
- `-results`: path to folder which will contain dumped test sources (prm/...), the default is the current folder.

Interpreting results

TODO: add example of how to interpret test output (both standard and detailed).

Test names

Every test has a unique identifier: test name. A test name consists of several words in lower case separated by slash (`/`). An example of test name is `prm/core/special/dispatchpacket/dim/basic/kernel_1_200x1x1_64x1x1_ND`. The following scheme for the words is used (starting from the beginning of test name):

- `prm`: Test suite
- `core`: Core PRM specification
- `special`: Chapter in PRM specification
- `dispatchpacket`: Section in PRM specification
- `dim`: Instruction or group of instructions in PRM specification
- `basic`: Name of test/scenario
- `kernel_1_200x1x1_64x1x1_ND`: Name/identifier of testcase within test. For `dim/basic` test, it is the testcase for dim instruction in kernel body with dispatch dimension `1`, grid sizes `200x1x1`, workgroup sizes `64x1x1` and no control directives.

The naming scheme for testcases depends on test. For many tests, it contains text representation of parameters described in Test details, separated by underscore (_). For some tests, simple counter is used. Refer to documentation below for the list of parameters.

A prefix of test names identifies a collection of tests that have names starting with this prefix. For example, `prm/core/special/dispatchpacket/` are the tests for dispatch packet operations.

Test parameters

Code Location: location of validated code

- Used sets:
- `All`: kernel, function
- `Kernel`: just kernel

Grid Geometry: geometry of a dispatch

- Consists of:
  - Number of dimensions
  - Grid sizes for `x`, `y`, `z`. Unused dimensions have value `1`.
  - Workgroup sizes for `x`, `y`, `z`. Unused dimensions have value `1`.
- Used sets:
  - `Trivial`: one work-item/work-group (dimension 1)

    - `All`: various geometries, includes samples of partial workgroups/workitems, corner cases
    - `DimensionSet`: representative geometry for each dimension
    - `OneGroupSet`: one group, one dimension geometry
    - `DefaultPlusNGroupSet`: set of geometry with 1 WorkItem, OneGroupSet, and 1 dim & N groups
    - `Boundary32Set`: samples of geometry with sizes `> 2^32`

    - `Boundary24Set`: samples of geometry with sizes `> 2^24`
    - `DegenerateSet`: samples of degenerated geometry when a dimension is used, but has size `1`

Control Directives: control directives in BRIG modules

- Control directives may be controlled with the following settings:
- Location: location of control directives (function, kernel or module).
- List of enabled control directives.
- A test shall use all possible combinations of enabled control directives. For example, if two directives are eabled - `requiredgridsize` and `requiredworkgroupsize`, the test shall use the following combinations:
    - `[]` (no control directives)
    - `[ requiredgridsize ]`
    - `[ requiredworkgroupsize ]`
    - `[ requiredgridsize, requiredworkgroupsize ]`
- Used sets:
    - `DimensionSet`: enabled directive is `requireddim` (directives affecting dimensions)
    - `GeometrySet`: enabled directives are `requireddim`, `requiredgridsize`, `requiredworkgroupsize` (directives affecting grid geometry)
    - `GridSizeSet`: enabled directives are `requireddim`, `requiredgridsize` (directives affecting grid size)
    - `DegenerateSet`: enabled directives are `requiredgridsize`, `requiredworkgroupsize` (directives affecting computation of operations that can be simplified for degenerate set)

Segment: memory segment

- Used sets:
    - `Atomic`: flat, global, group
    - `HasFlatAddress`: global, group, private (segments which may be accessed via flat address)

  - `MemFence`: global, group
  - `Variable`: global, readonly, kernarg, group, private, spill, arg
  - `All`: all standard segments

Operand Kind: kind of instruction operand

- Used sets:
  - `All`: register, immediate, WAVESIZE

Dst Type: type of destination operand

- Used sets:
  - `All`: all supported types

Src Type: type of source operands

- Used sets:
  - `All`: all supported types

Ftz: flush to zero modifier

- Used sets:
  - `All`: all legal values

Rounding: rounding modifier

- Used sets:
  - `All`: all supported values

Packed Controls: controls for processing packed operands

- Used sets:
  - `All`: all supported values

Compare Operation: operation used by cmp instruction

- Used sets:
  - `All`: all operations

Atomic Operation: operation used by atomic instructions

- Used sets:
  - `All`: all operations

Align: alignment

- Used sets:
  - `All`: all legal values

Const: constant memory access

- Used sets:
  - `All`: all legal values

Equiv: equivalence class

- Used sets:
  - `All`: all legal values

Memory Order

- Used sets:
  - `All`: rlx, screl, scacq, scar
  - `SignalAll`: rlx, screl, scacq, scar
  - `SignalWait`: rlx, screl, scacq
  - `MemFence`: screl, scacq, scar

Memory Scope

- Used sets:
  - `All`: wi, wv, wg, cmp, sys
  - `Global`: wv, wg, cmp, sys
  - `Group`: wv, wg
  - `Image`: wi, wv, wg, cmp, sys

Width: width modifier

- Used sets:
  - `UpToWavesizeAndAll`: 1, 2, 4, 8, 16, 32, WAVESIZE, All
  - `All`: all legal values

Test Data: data for testing arithmetic and memory operations

- Used sets:
  - `Standard`:
    - Includes the following values:
      - regular values;
      - boundary values (e.g. -128 and 127 for s8);
      - special values (NaNs, infinities, subnormals, etc);
      - values producing regular, boundary or special values as a result.
    - Does not include the following values:
      - values which signal exceptions;
      - values which result in an undefined behavior.

Test details

arithmetic: Arithmetic operations [chapter 5]

intfp: Integer/floating point arithmetic operations [chapters 5.2 and 5.11]

- Instructions: abs, add, borrow, carry, div, max, min, mul, mulhi, neg, rem, sub, ceil, floor, fma, fract, rint, sqrt, trunc
- Operand Kind: All
- Dst Type: All
- Ftz: All
- Rounding: All
- Packed controls: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

intopt: Integer optimization operations [chapter 5.3]

- Instructions: mad
- Operand Kind: All
- Dst Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

24int: 24-bit integer optimization operations [chapter 5.4]

- Instructions: mad24, mad24hi, mul24, mul24hi
- Operand Kind: All
- Dst Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

intshift: Integer shift operations [chapter 5.5]

- Instructions: shl, shr
- Operand Kind: All
- Dst Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

indbit: Individual bit operations [chapter 5.6]

- Instructions: and, or, xor, not, popcount
- Operand Kind: All
- Dst Type: All
- Src Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

bitstr: Bit string operations [chapter 5.7]

- Instructions: bitextract, bitinsert, bitask, bitrev, bitselect, firstbit, lastbit
- Operand Kind: All

- Dst Type: All
- Src Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

copymove: Copy and move operations [chapter 5.8]

- Instructions: combine, expand, mov
- Operand Kind: All
- Dst Type: All
- Src Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

packed: Packet data operations [chapter 5.9]

- Instructions: shuffle, unpacklo, unpackhi, pack, unpack
- Operand Kind: All
- Dst Type: All
- Src Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

bitcmov: Bit conditional move operation [chapter 5.10]

- Instructions: cmov
- Operand Kind: All
- Dst Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

fpbit: Floating-point bit operations [chapter 5.12]

- Instructions: class, copysign
- Operand Kind: All
- Dst Type: All
- Src Type: All

- Packed controls: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

nativefp: Native floating-point operations [chapter 5.13]

- Instructions: nsin, ncos, nlog2, nexp2, nsqrt, nrsqrt, nrcp, nfma
- Operand Kind: All
- Dst Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

multimedia: Multimedia operations [chapter 5.14]

- Instructions: bitalign, bytealign, lerp, packcvt, unpackcvt, sad, sadhi
- Operand Kind: All
- Dst Type: All
- Src Type: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

compare: Compare operation [chapter 5.17]

- Instructions: cmp
- Operand Kind: All
- Dst Type: All
- Src Type: All
- Compare Operation: All
- Ftz: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

conversion: Conversion operation [chapter 5.18]

- Instructions: cvt
- Operand Kind: All
- Dst Type: All

- Src Type: All
- Rounding: All
- Ftz: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

address: Address operations [chapters 5 and 11]

null: Verify result of address operation for null address obtained with nullptr [chapter 11.4]

- Instructions: stof, ftos, segmentp
- Segment: HasFlatAddress

identity: Verify converted address accesses same location as address before conversion [chapter 5.16]

- Instructions: stof, ftos
- Segment: HasFlatAddress
- segmentStore: use segment store/flat load and not vice versa
- nonull: use nonull in the instruction

variable: Verify result of segmentp operation for flat address pointing into a variable [chapter 5.15]

- Instructions: segmentp
- Segment: HasFlatAddress
- nonull: use nonull in the instruction

lda/alignment: Verify result of lda operation is divisible by alignment [chapter 5.8]

- Instructions: lda
- Segment: HasFlatAddress
- nonull: use nonull in the instruction

memory: Memory operations [chapter 6]

memory/ordinary: Ordinary memory operations [chapters 6.3 and 6.4]

- Instructions: ld, st
- Operand Kind: All
- Dst Type: All
- Segment: All
- Align: All
- Const: All
- Equiv: All
- Width: All
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

memory/atomic: Atomic and atomicnoret memory operations [chapter 6.6 and 6.7]

- Instructions: atomic, atomicnoret
- Operand Kind: All
- Dst Type: All
- Atomic operation: All
- Segment: Atomic
- Memory Order: All
- Memory Scope: All
- Equiv: all
- Grid Geometry: OneGroupSet
- Code Location: Kernel
- Test Data: Standard

memory/signal: Notification operations [chapter 6.8]

- Instructions: signal, signalnoret
- Memory Order: SignalAll, SignalWait

memfence: Memory fence operation [chapter 6.9]

- Instructions: memfence
- Grid Geometry: OneGroupSet

- Segment: Memfence
- Memory Order: Memfence
- Memory Scope: Global, Group, Image

Image operations [chapter 7]

Tests to be implemented.

Branch operations [chapter 8]

basic/br: Basic unconditional jump, verify expected result by setting value of HSAIL register [chapter 8]

- Instructions: br
- Code Location: All

basic: Basic conditional jump, switch conditional jump [chapter 8]

- Instructions: cbr, sbr
- Grid Geometry: DefaultPlusNGroupSet
- Width: UpToWavesizeAndAll
- Operand Kind: All

nested: Nested control flow (if-then-else) [chapter 8]

- Instructions: cbr, br
- Grid Geometry: DefaultPlusNGroupSet
- Width: UpToWavesizeAndAll
- Operand Kind: All

sand, sor: Short-circuit control flow [chapter 8]

- Instructions: cbr
- Grid Geometry: DefaultPlusNGroupSet
- Width: UpToWavesizeAndAll
- Operand Kind: All

Parallel Synchronization and communication operations [chapter 9]

barrier: Barrier operations [chapter 9.1] *More tests to be implemented.*

Fine-grain barrier operations [chapter 9.2]

basic: Basic fbarrier in divergent control flow.

- Used instructions: initfbar, joinfbar, waitfbar, leavefbar, releasefbar
- Grid Geometry: ??

example1: Use leavefbar to create an fbarrier that only contains divergent work-items (based on Example 1 from the specification)

- Used instructions: initfbar, joinfbar, waitfbar, leavefbar, releasefbar
- Grid Geometry: ??

example2: Use joinfbar to create an fbarrier that only contains divergent work-items (based on Example 2 from the specification)

- Used instructions: initfbar, joinfbar, waitfbar, leavefbar, releasefbar
- Grid Geometry: ??

example3: Producer/consumer using two fbarriers that allow producer and consumer wavefront executions to overlap (based on Example 3 from the specification)

- Used instructions: initfbar, joinfbar, waitfbar, arrivefbar, leavefbar, releasefbar
- Grid Geometry: ??

ldf: fbarrier operations on barrier address obtained with ldf

- Used instructions: initfbar, joinfbar, waitfbar, arrivefbar, leavefbar, releasefbar
- Grid Geometry: TrivialGeometrySet

joinleave_wait: Concurrent joinfbar+leavefbar and waitfbar operations

- Grid Geometry: ??

wait_arrive: Concurrent waitfbar and arrivefbar operations

- Grid Geometry: ??

wait_leave: Concurrent waitfbar and leavefbar operations

- Grid Geometry: ??

arrive_leave: Concurrent arrivefbar and leavefbar operations

- Grid Geometry: ??

wait_race: waitfbar in a loop

- Grid Geometry: ??

crosslane: Cross-lane operations [chapter 9.4] *More tests to be implemented.*

Function operations [chapter 10]

Direct call operation [chapter 10.6]

arguments: Verify passing argument/returning result of given type [chapter 10.2]

- Types: all BRIG types

More tests to be implemented.

special: Special operations [chapter 11]

dispatchpacket: Dispatch packet operations [chapter 11.1]

basic: Verify result of dispatch packet operation

- Instructions: currentworkgroupsize, dim, gridgroups, gridsize, workgroupid, workgroupsize, workitemabsid, workitemflatabsid, workitemflatid, workitemid
- Code Location: All
- Grid Geometry: DimensionSet for dim, All for others
- Control Directives: DimensionSet for dim, GeometrySet for others
- Type: u32/u64 for gridsize, workitemflatabsid, workitemflatid, u32 for others

boundary32: Verify result of dispatch packet operation for workitems with workitemflatabsid around 2ˆ32 boundary

- Instructions: gridsize, workitemflatabsid, workitemflatid
- Code Location: All
- Grid Geometry: Boundary32Set
- Control Directives: GridSizeSet
- Type: u32/u64

boundary24: Verify result of dispatch packet operation for possible mul24 finalizer optimizations around 2ˆ24 boundary

- Instructions: workitemabsid, workitemflatabsid, workitemflatid
- Code Location: All
- Grid Geometry: Boundary24Set
- Control Directives: Boundary24Set
- Type: u32/u64 for workitemflatabsid, workitemflatid, u32 for others

degenerate: Verify result of dispatch packet operation for used dimension that dispatched with size 1

- Instructions: currentworkgroupsize, gridgroups, gridsize, workgroupid, workgroupsize, workitemabsid, workitemflatid, workitemid
- Code Location: All
- Grid Geometry: DegenerateSet
- Control Directives: DegenerateSet
- Type: u32/u64 for workitemflatabsid, workitemflatid, u32 for others

packetid/basic: Compare result of packetid operation with value on the host

packetcompletionsig/basic: Compare result of packetcompletionsig operation with value on the host

Exception operations [chapter 11.2] *Tests to be implemented.*

usermodequeue: User mode queue operations [chapter 11.3]

basic: Verify result of queueid, queueptr, ldk operation for a dispatch of a kernel

- Instructions: queueid, queueptr, ldk
- Code Location: All

basicindex: Verify result of queue index operation on a user mode queue

- Instructions: ldqueuereadindex, ldqueuewriteindex, addqueuewriteindex, casqueuewriteindex, stqueuereadindex, stqueuewriteindex
- Code Location: All
- User mode queue type: separate, user-created

misc: Miscellaneous operations [chapter 11.4]

kernargbaseptr/identity: Verify accessing memory at kernargbaseptr address

- Code Location: All

kernargbaseptr/alignment: Verify alignment of kernargbaseptr depending on alignment of kernel arguments

- Code Location: All

nop: Verify kernel with nop instruction

- Code Location: All

cuid/lessmax: Verify that result of cuid operation is always less than maxcuid

- Code Locations: All
- Grid Geometry: All

cuid/identity: Verify that result of cuid operation is same across workgroup

- Code Locations: All
- Grid Geometry: All

maxcuid/identity: Verify that result of maxcuid operation is same across grid

- Code Locations: All
- Grid Geometry: All

clock/monotonic: Verify that result of clock operation increases monotonically

waveid/lessmax: Verify that result of waveid operation is always less than maxwaveid

- Code Locations: All
- Grid Geometry: All

waveid/identity: Verify that result of waveid operation is same across wavefront

- Code Locations: All
- Grid Geometry: All

maxwaveid/identity: Verify that result of maxwaveid operation is same across grid

- Code Locations: All
- Grid Geometry: All

laneid/lessmax: Verify that result of laneid operation is always less than wavesize

- Code Locations: All
- Grid Geometry: All

laneid/sequence: Verify that result of laneid operation corresponds to work-items assigned to lanes in work-item flattened absolute ID order

- Code Locations: All
- Grid Geometry: All

Exceptions: [chapter 12]

Tests to be implemented.

Directives: [chapter 13]

extension: extension directive

names: Simple usage of one directive

pair: Pairs of extension directive.

loc: loc directive

location: different locations of the directive

pragma: pragma directive

location: different locations of the directive

control: control directives

exception/location: different locations of the directive

- Locations: beginning of kernel, function

exception/argument: different arguments of exception directives

maxdynamic/location: different locations of the directive

geometry/location: different locations of the directives

- Directives: maxflatgridsize, maxflatworkgroupsize, requireddim, requiredgridsize, requiredworkgroupsize, requirenopartialworkgroups

Note: the effect of control directives is also tested in special operations tests

Version: [chapter 14]

Tests to be implemented.

Libraries: [chapter 15]

Tests to be implemented.

Profiles: [chapter 16]

Configuration/tests to be implemented.

Limits: [appendix A]

registers: Register limits

c, s, q, d: Use number of registers reaching limit for give register type

- Grid Geometry: TrivialGeometry

sdq: Use number of s, d, q registers reaching limit combined

- Grid Geometry: TrivialGeometry

liveregisters: Live register limits

c, s, q, d: Use number of live registers reaching limit for give register type

- Grid Geometry: TrivialGeometry

sdq: Use number of s, d, q live registers reaching limit combined

- Grid Geometry: TrivialGeometry

equiv: Use number of equivalence classes reaching limit

- Instructions: ld, st, atomic/atomicnoret operations
- Grid Geometry: TrivialGeometry

identifiers: *Tests to be implemented.*

wgsize: Use work-group size reaching minimal limit (256)

wavesize: Check wavesize is within limits and is a power of 2

wgnumber: Use number of work-groups reaching limit

dims: Grid dimensions reaching limit

fbarnumber: *Tests to be implemented.*

group_memory_size: Use group segment memory reaching limit

private_memory_size: Use private segment memory reaching limit

kernarg_memory_size: Use kernarg segment memory reaching limit

arg_memory_size: Use arg segment variables reaching limit

Test suite internals

Information to be added when the suite is finalized.