BEDROCK ARCHITECTURE

Bedrock Target Intrinsics

C Compiler Builtins and Header Interfaces

Draft

CONTENTS

LIST OF TABLES

1 SCOPE AND STATUS

This document defines the compiler-owned Bedrock C target interface: builtin names, implementation-reserved header wrappers, source signatures, lowering requirements, availability and privilege constraints, and compiler-visible effects. It is the contract shared by a Bedrock-aware C frontend, optimizer, backend, and the target headers.

Architectural instruction behavior remains defined by the Bedrock ISA Specification. The baseline calling convention is defined by the Bedrock C ABI. This document specifies how source programs reach target facilities; it does not duplicate their architectural or language semantics.

Interface name: `bedrock-target-intrinsics`

Source language: ISO C with implementation-reserved target interfaces

2 COMPILER INTERFACE MODEL

2.1 Names and Ownership

The compiler builtin namespace begins with `__builtin_bedrock_`. Each **Name** entry in this document denotes that prefix followed by the displayed name. For example, `cpuid` denotes `__builtin_bedrock_cpuid`. The ordinary header wrapper uses the `__bedrock_` prefix with the same name unless its entry specifies a macro spelling.

All of these identifiers are implementation-reserved. A builtin is a compiler primitive, not an external function symbol. The target headers own wrapper declarations and umbrella include topology; this document owns their ABI-visible meaning.

2.2 Header Topology and Exposure

`<bedrockintrin.h>` is the public umbrella. It includes the core, memory, integer, and floating-point families. `<bedrocksystemintrin.h>` is the system umbrella for system-register, cache, and MMU facilities. Any family header may also be included directly.

Table 2-1. Target Intrinsic Header Families

Family	Header	Umbrella	Exposure
Cache-Management	`<bedrockcacheintrin.h>`	system	cache management
Core	`<bedrockcoreintrin.h>`	public	unprivileged core operations
Floating-Point	`<bedrockfpuintrin.h>`	public	floating-point environment, classification, and approximate operations
Integer	`<bedrockintegerintrin.h>`	public	Bedrock-specific integer operations
Memory	`<bedrockmemoryintrin.h>`	public	user ordering and access hints
MMU	`<bedrockmmuintrin.h>`	system	supervisor translation operations
System-Register	`<bedrocksysregintrin.h>`	system	architectural system-register access

2.3 Common Compilation Rules

Family wrappers are static inline functions unless an encoded immediate or type operand requires a macro. Every builtin listed by an enabled profile must be recognized by the compiler. If a required optional feature is disabled, compilation fails rather than emitting an unresolved call.

An encoded immediate must be an integer constant expression in the stated range. Header inclusion neither suppresses nor emulates architectural privilege checks. An operation is volatile only when its entry states an externally observable architectural effect. Compiler memory effects and hardware ordering are exactly those stated by the entry; one does not imply the other.

The **C interface** column uses `result(arguments)` shorthand. Exact declarations are in the named header; `u8`, `u16`, `u32`, and `u64` abbreviate the corresponding `uintN_t` types, and `result` denotes `__bedrock_query_result_t`. A clobber means a compiler memory clobber, while a named fence also has the hardware behavior defined by the ISA.

3 PUBLIC TARGET INTERFACE

3.1 Core Builtins

Header: `<bedrockcoreintrin.h>`

Table 3-1. Core Intrinsics

Name	C interface	Lowering	Availability, constraint, and effect
`breakpoint`	`void(void)`	`BKPT`	Any privilege; always raises the architectural breakpoint/debug exception.
`cpuid`	`u64(u64)`	`CPUID`	User allowed; reads discovery state and not C memory.
`rdpmc`	`u64(performance_counter)`	`RDPMC`	Unprivileged; ID is a constant in 0 through 65535; zero and unavailable IDs fault with `INVALID_SELECTOR`.
`read_status`	`u16(void)`	`RDSTATUS`	Unprivileged; reads `STATUS` and not C memory.
`relax`	`void(void)`	`RELAX`	Unprivileged PAUSE-like convergent hint; retires normally and permits a finite implementation-selected delay, including zero, before subsequent execution.
`trace`	`void(u32)`	`TRACE`	Unprivileged; marker is a constant in 0 through 65535; emits a marker when enabled and otherwise acts as a NOP.
`yield`	`void(void)`	`YIELD`	Any privilege; scheduling hint, not a compiler or memory barrier.

3.2 Memory Builtins

Header: `<bedrockmemoryintrin.h>`

Table 3-2. Memory Intrinsics

Name	C interface	Lowering	Availability, constraint, and effect
address_fence	void(void)	AFENCE	Full compiler memory clobber and hardware address fence.
nontemporal_store_u16	void(u16 *,u16)	MOVNT.W	One writable 16-bit object; one hinted store, neither volatile nor a fence.
nontemporal_store_u32	void(u32 *,u32)	MOVNT.L	One writable 32-bit object; one hinted store, neither volatile nor a fence.
nontemporal_store_u64	void(u64 *,u64)	MOVNT.Q	One writable 64-bit object; one hinted store, neither volatile nor a fence.
nontemporal_store_u8	void(u8 *,u8)	MOVNT.B	One writable byte; one hinted store, neither volatile nor a fence.
read_fence	void(void)	RFENCE	Compiler and hardware read-ordering barrier.
write_fence	void(void)	WFENCE	Compiler and hardware write-ordering barrier.

3.3 Integer Builtins

Header: <bedrockintegerintrin.h>

Table 3-3. Integer Intrinsics

Name	C interface	Lowering	Availability, constraint, and effect
clmul_u16	u16(u16,u16)	CLMUL.W	Pure; low 16 bits of the carryless product.
clmul_u32	u32(u32,u32)	CLMUL.L	Pure; low 32 bits of the carryless product.
clmul_u64	u64(u64,u64)	CLMUL.Q	Pure; low 64 bits of the carryless product.
clmul_u8	u8(u8,u8)	CLMUL.B	Pure; low 8 bits of the carryless product.

3.4 Floating-Point Builtins

Header: <bedrockfpuintrin.h>

Table 3-4. Floating-Point Intrinsics

Name	C interface	Lowering	Availability, constraint, and effect
facosa_f32	float(float)	FACOSA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
facosa_f64	double(double)	FACOSA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fasina_f32	float(float)	FASINA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fasina_f64	double(double)	FASINA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fatana_f32	float(float)	FATANA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.

Name	C interface	Lowering	Availability, constraint, and effect
fatana_f64	double(double)	FATANA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fatanha_f32	float(float)	FATANHA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fatanha_f64	double(double)	FATANHA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fclass_f32	u16(float)	FCLASS.S	FPU required; pure ten-class one-hot classification; does not update FFLAGS.
fclass_f64	u16(double)	FCLASS.D	FPU required; pure ten-class one-hot classification; does not update FFLAGS.
fcosa_f32	float(float)	FCOSA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fcosa_f64	double(double)	FCOSA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fcosha_f32	float(float)	FCOSHA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fcosha_f64	double(double)	FCOSHA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fetoxa_f32	float(float)	FETOXA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fetoxa_f64	double(double)	FETOXA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fetoxm1a_f32	float(float)	FETOXM1A.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fetoxm1a_f64	double(double)	FETOXM1A.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
flog10a_f32	float(float)	FLOG10A.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
flog10a_f64	double(double)	FLOG10A.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
flog2a_f32	float(float)	FLOG2A.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
flog2a_f64	double(double)	FLOG2A.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
flogna_f32	float(float)	FLOGNA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
flogna_f64	double(double)	FLOGNA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.

Name	C interface	Lowering	Availability, constraint, and effect
flognp1a_f32	float(float)	FLOGNP1A.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
flognp1a_f64	double(double)	FLOGNP1A.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fsina_f32	float(float)	FSINA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fsina_f64	double(double)	FSINA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fsincosa_f32	void(float,float *,float *)	FSINCOSA.S	FP and FPTRANSA required; writes sine through the first result pointer and cosine through the second; ISA accuracy, special-value, and floating-point event contracts apply to both results.
fsincosa_f64	void(double,double *,double *)	FSINCOSA.D	FP and FPTRANSA required; writes sine through the first result pointer and cosine through the second; ISA accuracy, special-value, and floating-point event contracts apply to both results.
fsinha_f32	float(float)	FSINHA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
fsinha_f64	double(double)	FSINHA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
ftana_f32	float(float)	FTANA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
ftana_f64	double(double)	FTANA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
ftanha_f32	float(float)	FTANHA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
ftanha_f64	double(double)	FTANHA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
ftentoxa_f32	float(float)	FTENTOXA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
ftentoxa_f64	double(double)	FTENTOXA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
ftwotoxa_f32	float(float)	FTWOTOXA.S	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
ftwotoxa_f64	double(double)	FTWOTOXA.D	FP and FPTRANSA required; ISA accuracy, special-value, and floating-point event contracts apply.
read_fflags	u16(void)	RDFFLAGS	Unprivileged; reads accrued floating-point exception flags.

Name	C interface	Lowering	Availability, constraint, and effect
`read_fstatus`	`u16(void)`	`RDFSTATUS`	Unprivileged; access to the floating-point environment.
`write_fflags`	`void(u16)`	`WRFFLAGS`	Unprivileged; reserved bits follow the ISA; writes accrued flags.
`write_fstatus`	`void(u16)`	`WRFSTATUS`	Unprivileged; reserved bits follow the ISA; changes the floating-point environment.

3.5 Approximate Transcendental Lowering Policy

A strict language-library operation shall not be lowered to an FPTRANSA instruction. A compiler may select FPTRANSA only when the source uses a separately defined approximate interface or an active compiler mode permits approximate mathematics. The selected instruction remains subject to its ISA accuracy, special-value, and floating-point exception-condition contract.

4 SYSTEM TARGET INTERFACE

The compiler emits the requested architectural operation; runtime privilege and policy checks remain active.

4.1 System-Register Builtins

Header: `<bedrocksysregintrin.h>`

Table 4-1. System-Register Intrinsics

Name	C interface	Lowering	Availability, constraint, and effect
`read_code_segment`	`u64(void)`	`RDSEG CS`	Unprivileged; reads the current CS image through the dedicated source-only encoding.
`read_control_register`	`u64(control_register)`	`RDCR`	Supervisor; selector is a constant in 0 through 65535 naming a defined register.
`read_segment_register`	`u64(segment_register)`	`RDSEG`	Unprivileged; selector is a compile-time `__bedrock_segment_register_t`.
`write_control_register`	`void(control_register,u64)`	`WRCR`	Supervisor; defined constant selector required; writes the register and clobbers memory.
`write_segment_register`	`void(segment_register,u64)`	`WRSEG`	Unprivileged; validates the SREG image, changes address interpretation, and fully clobbers memory.
`write_status`	`void(u16)`	`WRSTATUS`	Supervisor; reserved and privilege-controlled bits follow the ISA; compiler-visible side effect.

The system-register header defines `__bedrock_control_register_t` constants for every published control-register selector. The constants are names for the architectural selector values; they do not weaken the privilege, reserved-bit, or per-register validation performed by RDCR and WRCR.

It also defines the pure value macros `__BEDROCK_SEGMENT_IMAGE`, `__BEDROCK_SEGMENT_IMAGE_FOR_BASE`, and `__BEDROCK_SEGMENT_DISABLED`. They construct 64-bit segment images without reading or writing architectural state.

`__BEDROCK_SEGMENT_IMAGE(base_page,exponent,mantissa, bounds_only)` places the low 52, 5, and 6 bits of the first three operands in their architectural fields and normalizes `bounds_only` to zero or one. Each operand is evaluated once. A valid enabled image uses an exponent from 0 through 31 and a mantissa from 1 through 63. The macro only packs fields; it does not validate the resulting base, span, limit, or address canonicality.

`__BEDROCK_SEGMENT_IMAGE_FOR_BASE(base,...)` encodes $\lfloor$`(uint64_t)base`$/4096\rfloor$ in the base-page field. It discards the low 12 bits, so a misaligned operand names the containing page and does not preserve the original byte address. The macro imposes no alignment precondition and does not adjust a later offset to compensate for the discarded bits. `__BEDROCK_SEGMENT_DISABLED` is the canonical all-zero disabled image.

4.2 Cache-Management Builtins

Header: `<bedrockcacheintrin.h>`

All cache range operations evaluate the address once and obtain the maintenance granule from CPUID `CACHE_TOPOLOGY`. A zero length emits no cache-maintenance instruction. A nonzero, non-wrapping byte range selects every maintenance block that intersects the range and lowers to one block instruction per selected block. Each instruction translates its own block address. If a later instruction faults, blocks completed by earlier instructions remain completed.

For a nonzero length, the mathematical half-open range $[\mathrm{address}, \mathrm{address} + \mathrm{length})$ shall lie within the 64-bit address space. Otherwise the behavior of the call is undefined and no architectural cache-maintenance sequence is required. An implementation may diagnose a provably wrapping constant range. A valid range end is calculated mathematically, not modulo 2^{64}.

Table 4-2. Cache-Management Intrinsics

Name	C interface	Lowering	Availability, constraint, and effect
`flush_dcache`	`void(void *,size_t)`	`FLSHDCACHE`	Supervisor; applies FLSHDCACHE to every CPUID maintenance block intersecting the range and fully clobbers memory.
`invalidate_dcache`	`void(void *,size_t)`	`INVDCACHE`	Supervisor; applies INVDCACHE to every CPUID maintenance block intersecting the range and fully clobbers memory.
`invalidate_icache`	`void(void *,size_t)`	`INVICACHE`	Supervisor; applies local INVICACHE to every CPUID maintenance block intersecting the range, serializes instruction fetch, and fully clobbers memory.
`sync_cache`	`void(void *,size_t)`	`SYNCCACHE`	Supervisor; applies SYNCCACHE to every CPUID maintenance block intersecting the range, serializes instruction fetch, and fully clobbers memory.

Name	C interface	Lowering	Availability, constraint, and effect
`writeback_dcache`	`void(void *,size_t)`	`WRBKDCACHE`	Supervisor; applies WRBKDCACHE to every CPUID maintenance block intersecting the range and fully clobbers memory.

4.3 MMU Builtins

Header: `<bedrockmmuintrin.h>`

Table 4-3. MMU Intrinsics

Name	C interface	Lowering	Availability, constraint, and effect
`invalidate_asid`	`void(u16)`	`INVASID`	Supervisor; ASID is a constant in 0 through 65535 and the operation fully clobbers memory.
`invalidate_page`	`void(const void *)`	`INVPAGE`	Supervisor; argument supplies a linear address and invalidates any matching cached leaf mapping containing it; the operation fully clobbers memory.
`invalidate_tlb`	`void(void)`	`INVTLB`	Supervisor; translation-state side effect and full memory clobber.
`page_table_query`	`query_result(u32,u64)`	`PTQUERY`	Supervisor; level is an integer constant expression in 1 through 5; wrapper atomically returns value and `FLAGS`; page walk without access to the queried C object.
`switch_page_table`	`void(u64)`	`SWPT`	Supervisor; valid page-table image required; changes address space and fully clobbers memory.
`switch_page_table_asid`	`void(u64,u16)`	`SWPTA`	Supervisor; valid image and low 16-bit ASID; changes address space and fully clobbers memory.
`virtual_to_physical`	`query_result(u64)`	`VTOP`	Supervisor; wrapper returns value and operation-defined `FLAGS` captured atomically; page walk without access to the translated C object.

5 SHARED HEADER TYPES

Table 5-1. Target Intrinsic Shared Types

Type	ABI contract
`__bedrock_control_register_t`	Architectural control-register selectors.
`__bedrock_performance_counter_t`	Architectural performance-counter selectors.
`__bedrock_query_result_t`	Value and architectural flags returned by MMU queries.
`__bedrock_segment_register_t`	Source-selectable architectural segment registers.