Intel® XScale™ Microarchitecture

Technical Summary

Product Features

- 7-8 stage Intel® Superpipelined RISC Technology achieves high speed and ultra low power
- Intel® Dynamic Voltage Management. Dynamic voltage & frequency on-the-fly scaling allows applications to utilize the right blend of performance and power
- Intel® Media Processing Technology. Multiply-accumulate coprocessor performs two simultaneous 16-bit SIMD multiplies with 40-bit accumulation for efficient media processing
- Power management unit gives power savings via idle, sleep and quick wake-up modes.
- 128-entry branch target buffer keeps pipeline filled with statistically correct branch choices
- 32 KB instruction cache keeps local copy of important instructions to enable high performance and low power
- 32 KB data cache keeps local copy of important data to enable high performance and low power
- 2 KB mini-data cache avoids “thrashing” of the d-cache for frequently changing data streams
- 32-entry instruction memory management unit enables logical-to-physical address translation, access permissions, i-cache attributes
- 32-entry data memory management unit enables logical-to-physical address translation, access permissions, d-cache attributes
- 4-entry fill and pend buffers promotes core efficiency by allowing “hit-under-miss” operation with data caches
- Performance monitoring unit furnishes two 32-bit event counters and one 32-bit cycle counter for analysis of hit rates, etc.
- Debug unit uses hardware breakpoints and 256-entry trace history buffer (for flow change messages) to debug programs
- 32-bit coprocessor interface provides high performance interface between core and coprocessors
- 64-bit core memory bus with simultaneous 32-bit input path and 32-bit output path gives up to 4.8 GBytes/sec. @ 600 MHz bandwidth for internal accesses
- 8-entry write buffer allows the core to continue execution while data is written to memory

Contents

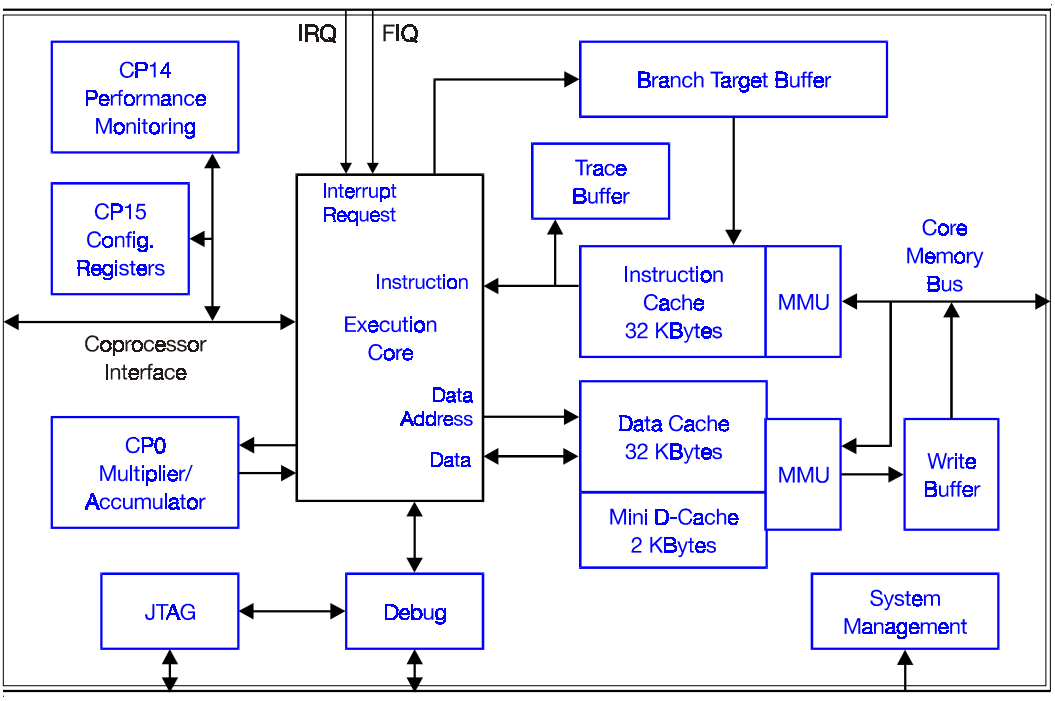
Tables

1.0 Introduction

The Intel® XScale™ microarchitecture is based on a new core which is compliant with ARM* version 5TE. The microarchitecture surrounds the core with instruction and data memory management units; instruction, data, and mini-data caches; write, fill, pend, and branch target buffers; power management, performance monitoring, debug, and JTAG units; coprocessor interface; 32K caches; MMUs; BTB; MAC coprocessor; and core memory bus.

The Intel XScale microarchitecture will be combined with peripherals to provide applications specific standard products (ASSP) targeted at selected market segments. As an example, the RISC core can be integrated with peripherals such as an LCD controller, multi-media controllers and an external memory interface to empower OEMs to develop smaller, more cost-effective handheld devices with long battery life, with the performance to run rich multimedia applications. As another example, the microarchitecture could be surrounded by high-bandwidth PCI interfaces, memory controllers and networking microengines to provide a highly integrated, low power, I/O or network processor.

The following block diagram shows the internal structure to the Intel XScale microarchitecture.



2.0 Programming Model

The Intel® Xscale™ microarchitecture uses the ARM* Version 5TE ISA programming model which handles 8-, 16-, and 32-bit data types and operates in one of seven processor modes: user, system, supervisor, abort, undefined instruction, fast interrupt, and normal interrupt. The microarchitecture provides sixteen general 32-bit registers (R0-R15) where R13 is the stack pointer (SP), R14 is the link register (LR), and R15 is the program counter (PC). It also supplements it's sixteen general registers plus a current program status register (CPSR) with 20 mode-dependent "shadow" registers as shown below.

Table 1. Mode-Dependent "Shadow" Registers

User	System	Supervisor	Abort	Undefined	Normal Int.	Fast Int.
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8
R9	R9	R9	R9	R9	R9	R9
R10	R10	R10	R10	R10	R10	R10
R11	R11	R11	R11	R11	R11	R11
R12	R12	R12	R12	R12	R12	R12
SP	SP	SP_SVC	SP_Abort	SP_Undef	SP_IRQ	SP_FIQ
LR	LR	LR_SVC	LR_Abort	LR_Undef	LR_IRQ	LR_FIQ
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_SVC	SPSR_Abort	SPSR_Undef	SPSR_IRQ	SPSR_IFQ

The "shadow" registers for the supervisor, abort, undefined instruction, normal interrupt (IRQ), and fast interrupt (FIQ) modes provide fast context switching by precluding the need to first save general registers. Upon entry into either of the supervisor, abort, undefined instruction, IRQ, or FIQ modes, the CPSR is "saved" in the mode-related saved program status register (SPSR).

The user mode and system mode use the same registers. Unlike the user mode, the system mode has privileges that allow it to the change the CPSR to select processor mode, turn on/off normal and fast interrupts, and select the thumb instruction set.

The core can handle these exception types:

Table 2. Exception Types

Exception Type	Vector Address	Mode
Reset	0x0	Supervisor
Undefined Instructions	0x4	Undefined
Software Interrupts	0x8	Supervisor
Prefetch Abort	0xC	Abort
Data Abort	0x10	Abort
Normal Interrupt	0x18	Interrupt
Fast Interrupt	0x1C	Fast Interrupt

The core supports vector address locations at either 0x0000,0000 or 0xFFFF,0000. However, the reset vector is always at address 0x0. Selection is made via coprocessor 15, register 1.

3.0 32-bit (ARM*) and 16-bit (Thumb*) Instruction Sets

The “T” in “ARM version 5TE ISA” means “Thumb” instruction set. The ARM version 5TE ISA executes either a 32-bit ARM instruction set or a 16-bit Thumb instruction set. The ARM instruction set is the default, the Thumb instruction set can be selected via the current program status register.

4.0 ARM V5 “DSP” Additions To ARM Instruction Set

ARM version 5 DSP additions to the ARM instruction set are:

Table 3. ARM* V5 “DSP” Additions To ARM* Instruction Set

Instruction	Operation
SMLAxy	32 <= 16x16 + 32
SMLAWy	32 <= 32x16 + 32
SMLALxy	64 <= 16x16 + 64
SMULxy	32 <= 16x16
SMULWy	32 <= 32x16
QADD	Adds 2 registers & saturates result if overflow occurred
QDADD	Doubles & saturates 1st register, then adds to 2nd reg & saturates
QSUB	Subtracts 2 registers & saturates result if overflow occurred
QDSUB	Doubles & saturates 1st reg, then subtracts 2nd reg & saturates

5.0 ARM V5 “Other” Additions To ARM Instruction Set

ARM version 5 additions to the ARM instruction set are:

Table 4. ARM* V5 “Other” Additions To ARM* Instruction Set

Instruction	Operation
BKPT	Software Breakpoint
BLX	Branch with Link and Exchange (switch to/from Thumb instruction set)
CLZ	Count Leading Zeroes
LDM/LDR > PC	Load from Memory or Register > PC can cause transfer to Thumb
MAR	Move CP0 40-bit Accumulator to 2 general ARM registers
MRA	Move 2 general ARM registers to CP0 40-bit Accumulator
LDRD	Load 2 general ARM registers from memory
STRD	Store 2 general ARM registers to memory
PLD	Preload Line > Data Caches (abort before any exceptions are taken)

6.0 Big Endian, Little Endian

The microarchitecture supports both big endian and little endian. Selection is made via coprocessor 15, register 1.

7.0 Coprocessor 15 (CP15)

CP15 provides registers that identify or control operation of functions within the microarchitecture.

Table 5. Coprocessor 15 (CP15) Registers

Register	Access	Description
0	Read / Write-ignored	ID
0	Read / Write-ignored	Cache Type
1	Read / Write	Control
1	Read / Write	Auxiliary Control
2	Read / Write	Translation Table Base
3	Read / Write	Domain Access control
4	Unpredictable	Reserved
5	Read / Write	Fault Status
6	Read / Write	Fault Address
7	Read-unpredictable / Write	Cache Operations
8	Read-unpredictable / Write	TLB Operations
9	Read / Write	Cache Lock Down
10	Read / Write	TLB Lock Down
11-12	Unpredictable	Reserved
13	Read / Write	Process ID (PID)
14	Read / Write	Breakpoint Registers
15	Read / Write	Coprocessor Access

8.0 Coprocessor 14 (CP14)

CP 14 provides registers that identify or control operation of functions within the microarchitecture.

Table 6. Coprocessor 14 (CP14) Registers

Register	Access	Description
0	Read / Write	Performance Monitoring: Control Register
1	Read / Write	Performance Monitoring: Clock Counter
2	Read / Write	Performance Monitoring: Event Counter #1
3	Read / Write	Performance Monitoring: Event Counter #2
4-5	Unpredictable	Reserved
6	Read / Write	Core Clock Configuration Register
7	Read / Write	Power Mode Register
8	Read / Write	Software Debug: TX Register
9	Read / Write	Software Debug: RX Register
10	Read / Write	Software Debug: Debug Control and Status Register
11	Read / Write	Software Debug: Trace Buffer Register
12	Read / Write	Software Debug: Checkpoint 0 Register
13	Read / Write	Software Debug: Checkpoint 1 Register
14	Read / Write	Software Debug: TXRX Control Register
15	Unpredictable	Reserved

9.0 Superpipeline

The superpipeline is composed of integer, multiply-accumulate (MAC), and memory pipes. The integer pipe has seven stages: branch target buffer/fetch 1, fetch 2, decode, register file/shift, ALU execute, state execute, and integer writeback. The memory pipe has eight stages that use the first five stages of the integer pipe (BTB/fetch 1 through ALU execute) and then finish with memory stages data cache 1, data cache 2, and data cache writeback. The MAC pipe has six to nine stages that use the first four stages of the integer pipe (BTB/fetch 1 through register file/shift) and then finish with MAC stages MAC1, MAC2, MAC 3, MAC 4, and data cache writeback. The MAC pipe supports a data-dependent early terminate where stages MAC2, MAC3, and/or MAC4 are by-passed.

Deep pipes promote high instruction execution rates only if a means exists to successfully predict the outcome of branch instructions. The branch target buffer provides such a means.

10.0 Branch Target Buffer (BTB)

Each entry of the 128-entry BTB contains the address of a branch instruction, the target address associated with the branch instruction, and a previous history of the branch being taken or not taken. The history is recorded as one of four states: strongly taken, weakly taken, weakly not-taken, or strongly not-taken. The BTB can be enabled or disabled via coprocessor 15, register 1.

If the address of the branch instruction hits in the BTB and it's history is strongly or weakly taken, the instruction at the branch target address is fetched; if it's history is strongly or weakly not-taken, the next sequential instruction is fetched. In either case the history is updated.

Data associated with a branch instruction enters the BTB the first time that the branch is taken. This data enters the BTB in a slot that has a history of strongly not-taken (overwriting previous data if present).

Successfully predicted branches avoid any branch-latency penalties in the superpipeline. Unsuccessfully predicted branches result in a 4-5 cycle branch-latency penalty in the superpipeline. BTB predictions are successful the lion's share of the time.

11.0 Instruction Memory Management Unit (IMMU)

For instruction prefetches, the IMMU controls logical-to-physical address translation, memory access permissions, memory domain identifications, and attributes (governing operation of the Instruction Cache). The IMMU contains a 32-entry, fully associative instruction translation look-a-side buffer (ITLB) that has a round-robin replacement policy. ITLB entries 0-30 can be locked.

If an instruction prefetch misses in the ITLB, the IMMU invokes an automatic table-walk mechanism that fetches an associated descriptor from memory and loads it into the ITLB. The descriptor contains information for logical-to-physical address translation, memory access permissions, memory domain identifications, and attributes governing operation of the i-cache. The IMMU then continues the instruction prefetch by using the address translation just entered into the ITLB. If an instruction prefetch hits in the ITLB, the IMMU continues the prefetch using the address translation already resident in the ITLB.

Access permissions for each of up to sixteen memory domains can be programed. If an instruction prefetch is attempted to an area of memory in violation of access permissions, then the attempt is aborted and a prefetch abort is sent to the core for exception processing. The IMMU and DMMU can be enabled or disabled together.

12.0 Data Memory Management Unit (DMMU)

For data fetches, the DMMU controls logical-to-physical address translation, memory access permissions, memory domain identifications, and attributes (governing operation of the data cache or mini-data cache and write buffer). The DMMU contains a 32-entry, fully associative data translation look-a-side buffer (DTLB) that has a round-robin replacement policy. DTLB entries 0-30 can be locked.

If a data fetch misses in the DTLB, the DMMU invokes an automatic table-walk mechanism that fetches an associated descriptor from memory and loads it into the DTLB. The descriptor contains information for logical-to-physical address translation, memory access permissions, memory domain identifications, and attributes (governing operation of the d-cache or mini-data cache and write buffer). The DMMU then continues the data fetch by using the address translation just entered into the DTLB. If a data fetch hits in the DTLB, the DMMU continues the fetch using the address translation already resident in the DTLB.

Access permissions for each of up to sixteen memory domains can be programed. If a data fetch is attempted to an area of memory in violation of access permissions, then the attempt is aborted and a data abort is sent to the core for exception processing. The IMMU and DMMU can be enabled or disabled together.

13.0 Instruction Cache (I-Cache)

The i-cache can contain high-use multiple code segments or entire programs, allowing the core access to instructions at core frequencies. This prevents core stalls caused by multicycle accesses to external memory.

The 32-KByte i-cache is 32-Set/32-way associative, where each set contains 32-ways and each way contains a tag address, a cache line (eight 32-bit words and one parity bit per word) of instructions, and a line-valid bit. For each of the 32 sets, 0-28 ways can be locked. Unlocked ways are replaceable via a round robin policy.

The i-cache can be enabled/disabled. Attribute bits within the descriptors contained in the ITLB of the IMMU provide some control over an enabled i-cache.

If a needed line (eight 32-bit words) is not present in the i-cache, the line is fetched (critical word first) from memory via a two-level-deep fetch queue. The fetch queue allows the next instruction to be accessed from the i-cache, but only if its data operands do not depend on the execution results of the instruction being fetched via the queue.

14.0 Data Cache (D-Cache)

The d-cache can contain high-use data such as lookup tables and filter coefficients, allowing the core access to data at core frequencies. This prevents core stalls caused by multicycle accesses to external memory.

The 32-KByte d-cache is 32-Set/32-way associative, where each set contains 32-ways and each way contains a tag address, a cache line (32 bytes with one parity bit per byte) of data, two dirty bits (one for each of two 16-byte groupings in a line), and one valid bit. For each of the 32 sets, 0-28 ways can be locked, unlocked, or used as local SRAM. Unlocked ways are replaceable via a round robin policy.

The d-cache (together with the mini-data cache) can be enabled/disabled. Attribute bits within the descriptors contained in the DTLB of the DMMU provide significant control over an enabled d-cache. These bits specify cache operating modes such as read and write allocate, write-back, write-through, and d-cache versus mini-data cache targeting.

The d-cache (and mini-data cache) work with the load buffer and pend buffer to provide “hit-under-miss” capability that allows the core to access other data in the cache after a “miss” is encountered (see section on load buffer, pend buffer for more information). The data cache (and mini-data cache) works in conjunction with the write buffer for data that is to be stored to memory (see section on write buffer for more information).

15.0 Mini-Data Cache

The mini-data cache can contain frequently changing data streams such as MPEG video, allowing the core access to data streams at core frequencies. This prevents core stalls caused by multicycle accesses to external memory. The mini-data cache relieves the d-cache of data “thrashing” caused by frequently changing data streams.

The 2-KByte mini-data cache is 32-Set/2-way associative, where each set contains 2-ways and each way contains a tag address, a cache line (32 bytes with one parity bit per byte) of data, two dirty bits (one for each of two 16-byte groupings in a line), and a valid bit. The mini-data cache uses a round robin replacement policy, and it cannot be locked.

The mini-data cache (together with the d-cache) can be enabled/disabled. Attribute bits contained within a coprocessor register specify operating modes: write and/or read allocate, write-back, and write-through.

The mini-data cache (and d-cache) work with the load buffer and pend buffer to provide “hit-under-miss” capability that allows the core to access other data in the cache after a “miss” is encountered (see section on load buffer, pend buffer for more information). The mini-data cache (and d-cache) works in conjunction with the write buffer for data that is to be stored to memory (see section on write buffer for more information).

16.0 Fill Buffer (FB) and Pend Buffer (PB)

The 4-entry FB works with the core to hold non-cacheable loads until the bus controller can act on them. The FB and the 4-entry PB work with the d-cache and mini-data cache to provide “hit-under-miss” capability, allowing the core to seek other data in the Caches while “miss” data is being fetched from memory. The FB can contain up to four unique “miss” addresses (logical), allowing four “misses” before the core is stalled. The PB holds up to four addresses (logical) for additional “misses” to those addresses that are already are in the FB. A coprocessor register can specify draining of the fill and pend (and write) buffers.

17.0 Write Buffer (WB)

The WB holds data for storage to memory until the bus controller can act on it. The WB is 8-entries deep, where each entry holds 16 bytes. The WB is constantly enabled, and accepts data from the core, d-cache, or mini-data cache.

Coprocessor 15, register 1 specifies whether WB coalescing is enabled or disabled. If coalescing is disabled, stores to memory occur in program order regardless of the attribute bits within the descriptors located in the DTLB. If coalescing is enabled, the attribute bits within the descriptors located in the DTLB are examined to determine if coalescing is enabled for the destination region of memory. If coalescing is enabled in both CP15, R1 and the DTLB, then data entering the WB can coalesce with any of the 8-entries (16 bytes) and then be stored to the destination memory region, but possibly out of program order.

Stores to a memory region specified to be non-cacheable and non-bufferable by the attribute bits within the descriptors located in the DTLB will cause the core to stall until the store completes. A coprocessor register can specify draining of the write (and fill and pend) buffer.

18.0 Interrupts

The microarchitecture responds to normal (IRQ) and (FIQ) fast interrupts.

19.0 Multiply-Accumulate Coprocessor (CP0)

For efficient processing of audio media algorithms, CP0 provides 40-bit accumulation of 16x16, dual 16x16 (SIMD), and 16x32 signed multiplies. Special MAR and MRA instructions are implemented to move 40-bit accumulator to two core general registers (MAR) and move two core general registers to 40-bit accumulator (MRA).

16x16 signed multiply-accumulates (MIAxy) multiply either the high/high, low/low, high/low, or low/high 16-bits of a 32-bit core general register (multiplier) and another 32-bit core general register (multiplicand) to produce a full 32-bit product which is sign-extended to 40-bits and then added to the 40-bit accumulator.

Dual signed 16x16 (SIMD) multiply-accumulates (MIAPH) multiply the high/high and low/low 16-bits of a packed 32-bit core general register (multiplier) and another packed 32-bit core general register (multiplicand) to produce two 16-bits products which are both sign-extended to 40-bits and then both added to the 40-bit accumulator.

32x32 signed multiply-accumulates (MIA) multiply a 32-bit core general register (multiplier) and another 32-bit core general register (multiplicand) to produce a 64-bit product where the 40 LSBs are added to the 40-bit accumulator. 16x32 versions of the 32x32 multiply-accumulate instructions complete in a single cycle.

20.0 Coprocessor Interface

The coprocessor interface uses a 32-bit bus for data transfers (at core frequency) between a coprocessor and core registers. It supports both tightly coupled and parallel execution modes, where, in tightly coupled mode the core waits for the coprocessor to complete execution of a command, and in parallel mode the core issues a command to a coprocessor and the core continues executing other instructions while the coprocessor is executing the command (perhaps a series of coprocessor-based micro-instructions).

21.0 Internal Memory Bus

The internal memory bus can simultaneously transfer one 32-bit word of load data and one 32-bit word of store data every clock cycle. This corresponds to a maximum data rate (when the core can sustain it) of 2.4 GBytes/sec. in each direction at 600 MHz, and results in a 4.8 GBytes/sec. total data rate at 600 MHz. The 4.8-GBytes/sec. figure is derived as follows:

64-bits = 8-Bytes

4.8-GBytes/sec. divided by 8-Bytes = 0.6-G/sec.

0.6-G/sec. = 600 MHz, the maximum core clock frequency

22.0 Clock and Power Management

The core is designed with power saving techniques that power-up a functional block only when it is needed. Low power modes are selectable by programming CP 14, register 6. The core is specifically designed to enable dynamic clocking. The core's clock frequency is set by programming CP14, register 7. This enables software to conserve power by matching the core clock frequency to the current workload. Dynamic clocking also optimizes performance from 40-mW/185-MIPS at 150-MHz; 450-mW/750-MIPS at 600-MHz and up to 900-mW/1000-MIPS at 800 MHz.

23.0 Performance Monitoring Unit

The performance monitoring unit contains two 32-bit event counters and one 32-bit clock counter. The event counters can be programmed to monitor i-cache hit rate, data caches hit rate, ITLB hit rate, DTLB hit rate, pipeline stalls, BTB prediction hit rate, and instruction execution count. Up to eight additional events can be monitored when using the Intel® XScale™ microarchitecture as the basis for an application specific standard product (ASSP).

24.0 JTAG

The industry-standard IEEE1149.1 JTAG port consists of a test access port (TAP) controller, boundary-scan register, instruction and data registers, and dedicated signals TDI, TDO, TCK, TMS, and nTRST. The JTAG port can also be used to access the debug unit, which provides control over the debugging of program code and the monitoring of debug data.

25.0 Debug Unit

The debug unit is accessed through the JTAG port. The debug unit, when used with debugger application code running on a host system outside of the Intel XScale microarchitecture, allows a program running on the Intel XScale microarchitecture to be debugged. It allows the debugger application code or a debug exception to stop program execution and re-direct execution to a debug handling routine. Debug exceptions are instruction breakpoint, data breakpoint, software breakpoint, external debug breakpoint, exception vector trap, and trace buffer full breakpoint. Once execution has stopped, the debugger application code can examine or modify the core's state, co-processor state, or memory. The debugger application code can then restart program execution.

The debug unit has two hardware instruction breakpoint registers, two hardware data breakpoint registers, and a hardware data breakpoint control register. The second data breakpoint register can be alternatively used as a mask register for the first data breakpoint register. A 256-entry trace buffer provides the ability to capture control flow messages or addresses. A JTAG instruction (LDIC) can be used to download a debug handler via the JTAG port to the mini-instruction cache (the i-cache has a 2-KByte mini-instruction cache, like the mini-data cache, that is used only to hold a debug handler).