SECTION 5
INSTRUCTION CACHE

The instruction cache (I-cache) is a 4-Kbyte, 2-way set associative cache. The cache is organized into 128 sets, with two lines per set and four words per line. Cache lines are aligned on 4-word boundaries in memory.

A cache access cycle begins with an instruction request from the CPU instruction unit. In case of a cache hit, the instruction is delivered to the instruction unit. In case of a cache miss, the cache initiates a burst read cycle (four beats per burst, one word per beat) on the instruction bus (I-bus) with the address of the requested instruction. The first word received from the bus is the requested instruction. The cache forwards this instruction to the instruction unit as soon as it is received from the I-bus. A cache line is then selected to receive the data that will be coming from the bus. A least-recently-used (LRU) replacement algorithm is used to select a line when no empty lines are available.

Each cache line can be used as an SRAM, allowing the application to lock critical code segments that need fast and deterministic execution time.

Cache coherency in a multiprocessor environment is maintained by software and supported by a fast hardware invalidation capability.

5.1 Instruction Cache Organization

Figure 5-1 illustrates the I-cache organization.

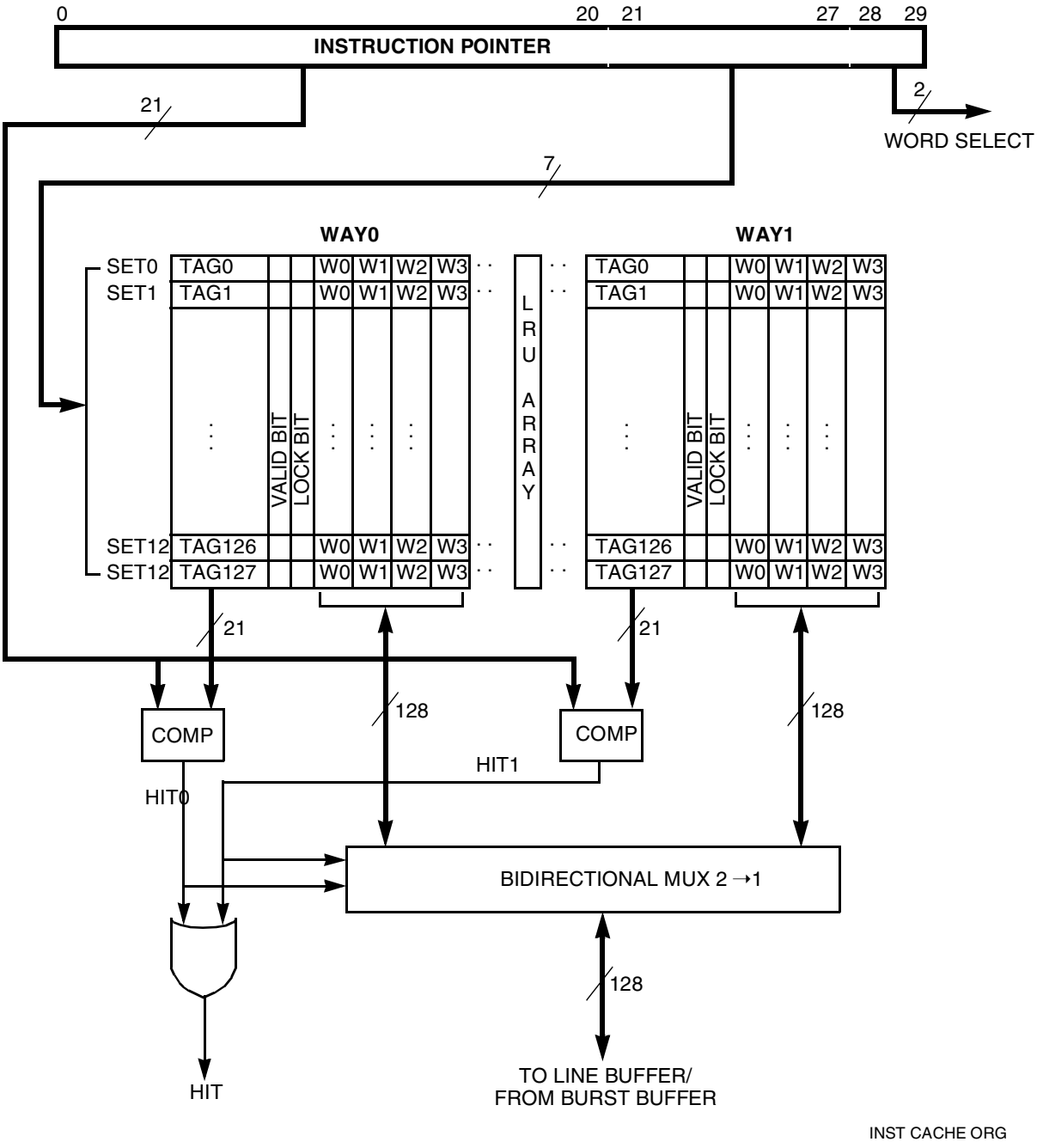

Figure 5-1 Instruction Cache Organization

Figure 5-2 illustrates the data path of the I-cache.

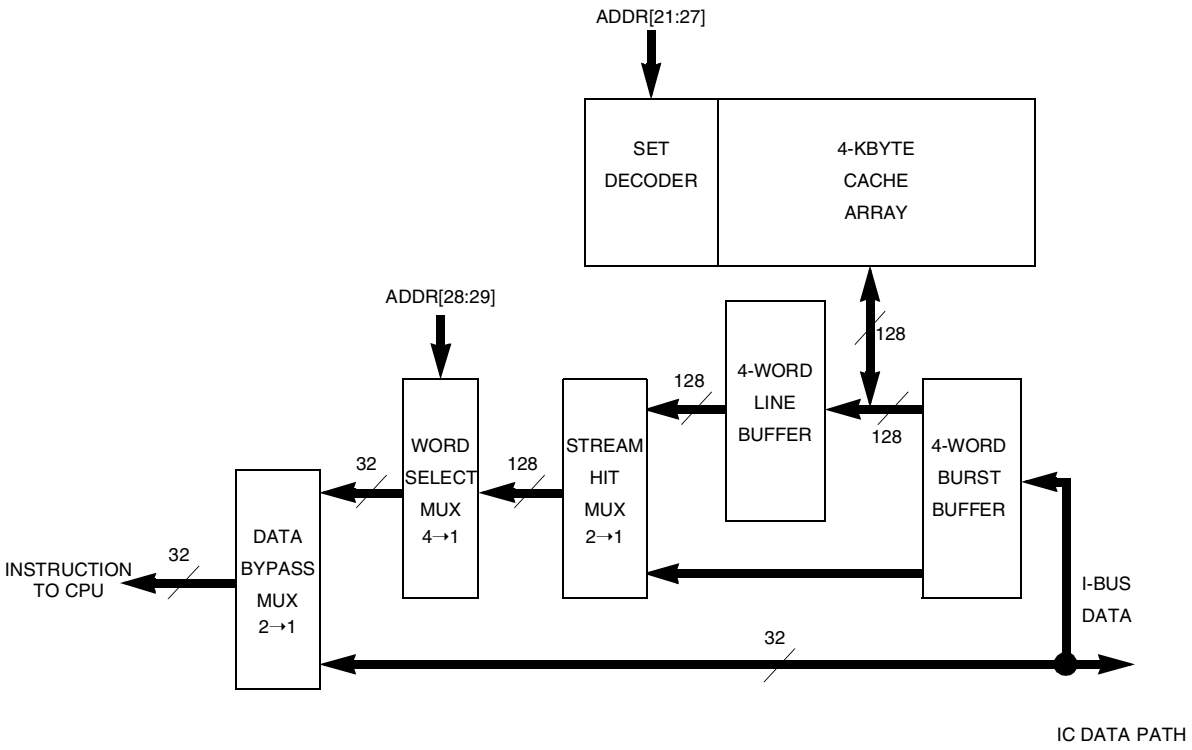

Figure 5-2 Instruction Cache Data Path

5.2 Programming Model

Table 5-1 lists the special purpose registers (SPRs) that control the operation of the I-cache.

Table 5-1 Instruction Cache Programming Model

SPR Number (Decimal)	Name	Description
560	ICCST	I-cache control and status register
561	ICADR	I-cache address register
562	ICDAT	I-cache data port (read only)

These registers are privileged; attempting to access them when the CPU is operating at the user privilege level results in a program interrupt.

5.2.1 I-Cache Control and Status Register (ICCST)

The ICCST contains control bits for enabling the I-cache and executing I-cache commands and status bits to indicate error conditions.

ICCST — I-Cache Control and Status Register **SPR 560**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
IEN	RESERVED			CMD			RESERVED			CCER 1	CCER 2	CCER 3	RESERVED		
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
RESERVED															
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5-2 ICCST Bit Settings

Bits	Mnemonic	Description
0	IEN	I-cache enable status bit. This bit is a read-only bit. Any attempt to write it is ignored 0 = I-cache is disabled 1 = I-cache is enabled
[1:3]	—	Reserved
[4:6]	CMD	I-Cache Command 000 = No command 001 = **Cache enable** 010 = **Cache disable** 011 = **Load & lock** 100 = **Unlock line** 101 = **Unlock all** 110 = **Invalidate all** 111 = Reserved
[7:9]	—	Reserved
10	CCER1	I-Cache Error Type 1 (sticky bit) 0 = No error 1 = Error
11	CCER2	I-Cache Error Type 2 (sticky bit) 0 = No error 1 = Error
12	CCER3	I-Cache Error Type 3 (sticky bit) 0 = No error 1 = Error
[13:31]	—	Reserved

5.2.2 I-Cache Address Register (ICADR)

Writing to the ICADR assigns the address that will be used by subsequent I-cache commands that are programmed in the ICCST.

ICADR — I-Cache Address Register **SPR 561**

0	31
ADR	

RESET: UNDEFINED

Table 5-3 ICADR Bit Settings

Bits	Mnemonic	Description
[0:31]	ADR	The address to be used in the command programmed in the control and status register

5.2.3 I-Cache Data Register (ICDAT)

The ICDAT register contains the data received when the I-cache tag array is read.

ICDAT — I-Cache Data Register **SPR 562**

0	31
DAT	

RESET: UNDEFINED

Table 5-4 ICDAT Bit Settings

Bits	Mnemonic	Description
[0:31]	DAT	The data received when reading information from the I-cache

5.3 Instruction Cache Operation

On an instruction fetch, bits 21 to 27 of the instruction's address are used as an index into the cache to retrieve the tags and data of one set. The tags from both accessed lines are then compared to bits 0 to 20 of the instruction's address. If a match is found and the matched entry is valid, then the access is a cache hit.

If neither tag matches or if the matched tag is not valid, the access is a cache miss.

The I-cache includes one burst buffer that holds the last line received from the bus, and one line buffer that holds the last line received from the cache array. If the requested data is found in one of these buffers, the access is considered a cache hit.

To minimize power consumption, the I-cache attempts to make use of data stored in one of its internal buffers. Using a special indication from the CPU, it is also possible, in some cases, to detect that the requested data is in one of the buffers early enough so the cache array is not activated at all.

5.3.1 Cache Hit

On a cache hit, bits 28 to 29 of the instruction's address are used to select one word from the cache line whose tag matched. In the same clock cycle, the instruction is transferred to the instruction unit of the processor.

5.3.2 Cache Miss

On a cache miss, the address of the missed instruction is driven on the I-bus with a four-word burst transfer read request. A cache line is then selected to receive the data that will be coming from the bus. The selection algorithm gives first priority to invalid lines. If neither of the two candidate lines in the selected set are invalid, then the least recently used line is selected for replacement. Locked lines are never replaced.

The transfer begins with the word requested by the instruction unit (critical word first), followed by any remaining words of the line, then by any remaining words at the beginning of the line (wrap around). As the missed instruction is received from the bus, it is immediately delivered to the instruction unit and also written to the burst buffer.

As subsequent instructions are received from the bus they are also written into the burst buffer and, if needed, delivered to the instruction unit (stream hit) either directly from the bus or from the burst buffer. When the entire line resides in the burst buffer, it is written to the cache array if the cache array is not busy with an instruction unit request.

If a bus error is encountered on the access to the requested instruction, a machine check exception is taken. If a bus error occurs on any access to other words in the line, the burst buffer is marked invalid and the line is not written to the array. If no bus error is encountered, the burst buffer is marked valid and eventually is written to the array.

Together with the missed word, an indication may arrive from the I-bus that the memory device is non-cacheable. If such an indication is received, the line is written only to the burst buffer and not to the cache. Instructions stored in the burst buffer that originated in a cache-inhibited memory region are used only once before being refetched. Refer to **5.4.8 Cache Inhibit** for more information.

5.3.3 Instruction Fetch on a Predicted Path

The processor implements branch prediction to allow branches to issue as early as possible. This mechanism allows instruction pre-fetch to continue while an unresolved branch is being computed and the condition is being evaluated. Instructions fetched following unresolved branches are said to be fetched on a predicted path.

These instructions may be discarded later if it turns out that the machine has followed the wrong path.

To minimize power consumption, the I-cache does not initiate a miss sequence in most cases when the instruction is inside a predicted path. The I-cache evaluates fetch requests to determine whether they are inside a predicted path. If a hit is detected, the requested data is delivered to the processor. However, on a cache miss, in most cases the cache-miss sequence is not initiated until the processor finishes the branch evaluation.

5.4 Cache Commands

The instruction cache supports the PowerPC instruction cache block invalidate (**icbi**) instruction together with some additional commands that help control the cache and debug the information stored in it. The additional commands are implemented using the three special purpose control registers ICCST, ICADR, and ICDAT.

Most of the commands are executed immediately after the control register is written and cannot generate any errors. When these commands are executed, there is no need to check the error status in the ICCST.

The **load & lock** command may take longer and may generate errors. When executing this command, the user needs to insert an **isync** instruction immediately after the I-cache command and check the error status in the ICCST after the **isync** instruction. The error type bits in the ICCST are sticky, allowing the user to perform a series of I-cache commands before checking the termination status. These bits are set by hardware and cleared by software.

Only commands that are not executed immediately need to be followed by an **isync** instruction for the hardware to perform them correctly. However, all commands need to be followed by **isync** in order to make sure all fetches of instructions that follow the I-cache command in the program stream are affected by the I-cache command.

Because the ICCST is a supervisor-level register, cache commands that require setting bits in this register are accessible only at the supervisor privilege level (MSR[PR] = 0). Attempting to write this register at the user privilege level results in a program exception.

The CPU **icbi** instruction (discussed below) can be performed at the user privilege level.

5.4.1 Instruction Cache Block Invalidate

The PowerPC instruction cache block invalidate (**icbi**) instruction invalidates the cache block indicated by the effective address in the instruction. The RCPU implements this instruction as if it pertains only to the on-chip instruction cache. This instruction does not broadcast on the external bus, and the RCPU does not snoop this instruction if broadcast by other masters.

This command is not privileged and has no error cases that the user needs to check.

The I-cache performs this instruction in one clock cycle. In order to calculate the latency of this instruction accurately, bus latency should be taken into account.

5.4.2 Invalidate All

To invalidate the whole cache, set the **invalidate all** command in the ICCST. This command has no error cases that the user needs to check.

When the command is invoked, if MSR[PR] = 0, all valid lines in the cache, except lines that are locked, are made invalid. As a result of this command, the LRU of all lines points to an unlocked way or to way zero if both lines are not locked. This last feature is useful in order to initialize the I-cache out of reset.

The I-cache performs this instruction in one clock cycle. In order to calculate the latency of this instruction accurately, bus latency should be taken into account.

5.4.3 Load and Lock

The **load & lock** operation is used to lock critical code segments in the cache. The **load & lock** operation is performed on a single cache line. After a line is locked it operates as a regular instruction SRAM; it will not be replaced during future misses and will not be affected by invalidate commands.

The following sequence loads and locks one line:

1. Read error type bits in the ICCST in order to clear them
2. Write the address of the line to be locked to the ICADR
3. Set the **load & lock** command in the ICCST
4. Issue the **isync** instruction
5. Return to step 2 to load and lock more lines
6. Read the error type bits in the ICCST to determine whether the operation completed properly

After the **load & lock** command is written to the ICCST, the cache checks if the line containing the byte addressed by the ICADR is in the cache. If it is, the line is locked and the command terminates with no exception. If the line is not in the cache a regular miss sequence is initiated. After the whole line is placed in the cache the line is locked.

The user needs to check the error type bits in the ICCST to determine if the operation completed properly or not. The **load & lock** command can generate two errors:

- Type 1 — bus error in one of the cycles that fetches the line
- Type 2 — no place to lock. It is the responsibility of the user to make sure that there is at least one unlocked way in the appropriate set.

5.4.4 Unlock Line

The **unlock line** operation is used to unlock locked cache lines. The **unlock line** operation is performed on a single cache line. If the line is found in the cache (cache hit), it is unlocked and starts to operate as a regular valid cache line. If the line is not found in the cache (cache miss), no operation is performed, and the command terminates with no exception.

The following sequence unlocks one cache line:

1. Write the address of the line to be unlocked to the ICADR
2. Set the **unlock line** command in the ICCST

This command has no error cases that the user needs to check.

The I-cache performs this instruction in one clock cycle. To calculate the latency of this instruction accurately, bus latency should be taken into account.

5.4.5 Unlock All

The **unlock all** operation is used to unlock the whole cache. This operation is performed on all cache lines. If a line is locked it is unlocked and starts to operate as a regular valid cache line. If a line is not locked or if it is invalid no operation is performed.

In order to unlock the whole cache set the **unlock all** command in the ICCST.

This command has no error cases that the user needs to check.

The I-cache performs this instruction in one clock cycle. To calculate the latency of this instruction accurately, bus latency should be taken into account.

5.4.6 Cache Enable

To enable the cache, set the **cache enable** command in the ICCST. This operation can be performed only at the supervisor privilege level. The **cache enable** command has no error cases that the user needs to check.

5.4.7 Cache Disable

To disable the cache, set the **cache disable** command in the ICCST. This operation can be performed only at the supervisor privilege level. The cache disable command has no error cases that the user needs to check.

5.4.8 Cache Inhibit

A memory region can be programmed in the chip select logic to be cache inhibited. When an instruction is fetched from a cache-inhibited region, the full line is brought to the internal burst buffer. Instructions stored in the burst buffer that originated from a cache-inhibited region may be sent to the processor no more than once before being re-fetched.

When changing a memory region (by writing to the appropriate chip-select registers) from a cache-enabled to a cache-inhibited region, the user must do the following:

1. Unlock all locked lines containing code that originated in this memory region
2. Invalidate all lines containing code that originated in this memory region
3. Execute an **isync** instruction

If these steps are not followed, code from a cache-inhibited region could be left inside the cache, and a reference to a cache-inhibited region could result in a cache hit. When a reference to a cache-inhibited region results in a cache hit, the data is delivered to the processor from the cache, not from memory.

When the FREEZE signal is asserted, indicating that the processor is under debug, all fetches from the cache are treated as if they were from cache-inhibited regions.

5.4.9 Cache Read

The user can read all data stored in the I-cache, including the data stored in the tags array.

To read the data that is stored in the I-cache,

1. Write the address of the data to be read to the ICADR. Note that it is also possible to read this register for debugging purposes.
2. Read the ICDAT

So that all parts of the I-cache can be accessed, the ICADR is divided into the following fields:

Table 5-5 ICADR Bits Function for the Cache Read Command

[0:17]	18	19	20	[21:27]	[28:29]	[30:31]
Reserved	0 = tag 1 = data	0 = way 0 1 = way 1	Reserved	Set select	Word select (used only for data array)	Reserved

When the data array is read from, the 32 bits of the word selected by the ICADR are placed in the target general-purpose register.

When the tag array is read, the 21 bits of the tag selected by the ICADR, along with additional information, are placed in the target general-purpose register. **Table 5-6** illustrates the bits layout of the I-cache data register when a tag is read.

Table 5-6 ICDAT Layout During a Tag Read

[0:20]	21	22	23	24	[25:31]
Tag value	Reserved	0 = not valid 1 = valid	0 = not locked 1 = locked	LRU bit	Reserved

5.5 I-Cache and On-Chip Memories with Zero Wait States

On-chip memories on the I-bus are considered to be cache-inhibited memory regions.

Performing a **load & lock** with such an on-chip memory is not advised. In most cases the instruction will still be fetched from the on-chip memory, even though it is also present in the I-cache.

5.6 Cache Coherency

Cache coherency in a multi-processor environment is maintained by software and supported by the invalidation mechanisms described in **5.4 Cache Commands**. All instruction storage is considered to be in “coherency not required” mode.

5.7 Updating Code and Attributes of Memory Regions

When updating code or when changing the attributes of memory regions (by writing to chip-select registers), the user must perform the following actions:

1. Update code or change memory region programming in the chip-select logic.
2. Execute the **sync** instruction to ensure the update or change operation finished.
3. Unlock all locked lines containing code that was updated.
4. Invalidate all lines containing code that was updated.
5. Execute the **isync** instruction.

5.8 Reset Sequence

To simplify system debugging, the I-cache is forced to be disabled only during reset (ICCST[EN] = 0). This feature enables the user to investigate the exact state of the I-cache prior to the event that asserted the reset.

In order to ensure proper operation of the I-cache after reset, the following actions must be performed:

1. **unlock all**
2. **invalidate all**
3. **cache enable**

5.9 Debugging Support

The processor can be debugged either in debug mode or by a software monitor debugger. In both cases the processor asserts the internal FREEZE signal. For a detailed description of RCPU debugging support, refer to **SECTION 8 DEVELOPMENT SUPPORT**.

When FREEZE is asserted, the I-cache treats all misses as if they were from cache-inhibited regions. That is, the misses are loaded only to the burst buffer and the cache state therefore remains exactly the same (assuming the debug routine is not in the I-cache). Notice that when FREEZE is asserted, cache hits are still read from the array, and therefore the LRU bits are updated.

5.9.1 Running a Debug Routine from the I-Cache

It may be desirable, in some cases, to be able to run a debug routine from the I-cache (e.g., for performance reasons). The following steps could be used to run the debug routine from the I-cache:

1. Save both ways of the sets that are needed for the debug routine by reading the tag value, LRU bit value, valid bit value, and lock bit value.
2. **Unlock** the locked ways in the selected sets.
3. Use **load & lock** to load and lock the debug routine into the I-cache (**load & lock** operates the same when FREEZE is asserted).
4. Run the debug routine. All accesses to it will result in hits.

After the debug routine is finished, the old state of the I-cache can be restored by following these steps:

1. **Unlock** and **invalidate** all the sets that are used by the debug routine (both ways).
2. Use **load & lock** to restore the old sets.
3. **Unlock** the ways that were not locked before.
4. In order to restore the old state of the LRU, make sure the last access (**load & lock** or **unlock**) is performed the MRU way (not the LRU way).

5.9.2 Instruction Fetch from the Development Port

When the processor is in debug mode, all instructions are fetched from the development port, regardless of the address generated by the processor. The I-cache is therefore bypassed in debug mode.