Xsight

Xsight Labs®

XISA

X-Switch Instruction Set Architecture

Document Status: Released
Document Number: TD-402-00
Revision: 1.00
Document Classification: Public
Date: March 16, 2025

Document Details

- Technical Publication: **Public**
- Document Status: **Released**
- Document Revision: 1.00

For more information contact us at www.xsightlabs.com

Table of Contents

1 Introduction

Xsight Lab's X-Switch family of devices features fully programmable switches, engineered to support the development of standard, emerging, and proprietary network protocols and applications.

X-Switch's open-to-the-public Instruction Set Architecture (ISA), the **X**$^{\textbf{ISA}}$ defines the foundational framework for managing packet processing within the X-Switch family of devices.
X$^{\textbf{ISA}}$ is designed to be generic, allowing users to develop and execute custom data plane programs. It enables writing programs running on a Packet Parser and Match-Action-Processing (MAP) units. Both the Parser and the MAP cores feature a comprehensive instruction set optimized for efficient packet header processing. By leveraging this architecture, developers can implement high-performance, scalable, and flexible network switching solutions tailored to diverse networking requirements.

At its core, **X**$^{\textbf{ISA}}$ enables advanced network programmability by providing a structured, yet flexible instruction set. The architecture is designed to enable packet processing at line rate, ensuring optimized performance for functions such as packet classification, forwarding, load balancing, tunneling and telemetry. Including a programmable Parser allows for flexible packet header parsing, while the MAP cores execute complex packet-handling tasks with efficiency. These components work together seamlessly, offering a powerful platform for data plane operations.

This document serves as a comprehensive guide for software and firmware developers seeking to integrate Xsight Labs' switches into their network infrastructure. It provides a detailed breakdown of all available instructions, their variations, and their intended applications. By following the guidelines outlined here, developers can write optimized code to fully leverage **X**$^{\textbf{ISA}}$ capabilities.

Beyond its utility for developers, this document will also aid network engineers and system architects when designing networks, enhance their optimization, and shorten troubleshooting time.

2 Programming Forwarding Engine (PFE)

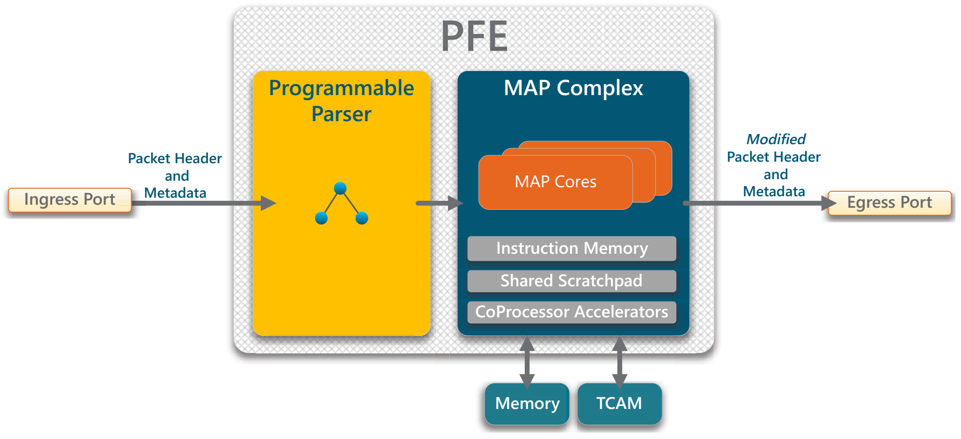
Overview

The X-Switch ISA, X^{ISA}, is executed by the Programmable Forwarding Engine (PFE), which integrates the Programmable Parser and the MAP cores. This section describes the main components of the PFE, as well as a high level description of the packet processing flow.

2.1 PFE Components

The following sub-sections describe the functions of the different PFE components illustrated in Figure 1.

Figure 1: PFE Main Components



2.1.1 Programmable Parser

The Parser is a software programmable engine with optimized resources and a specialized instruction set (Parser ISA) for parsing packet headers. As such, it offloads major parsing tasks from the MAP cores. The division of work between the Parser and the MAP remains, however, flexible and application-dependent.

2.1.2 Match-Action Processor (MAP) Complex

The MAP Complex includes the MAP cores, and other shared resources: Instruction Memory, Scratchpad Memory, and CoProcessor Accelerators. The MAP Complex is optimized for performing data plane packet processing tasks such as security, classification, forwarding, and header editing.

The MAP core is a processor supporting an instruction-set (MAP ISA) optimized to perform packet processing tasks and general purpose arithmetic and memory access. Packets are assigned to a MAP core, which processes the packets independently in a run-to-completion mode. The instruction memory and scratchpad memory are shared by all MAP cores. The CoProcessor accelerators also shared by all MAP cores, accelerate various common packet processing tasks.

2.2 Packet Header Processing Flow

The PFE packet header processing flow, depicted in Figure 2 involves three stages:

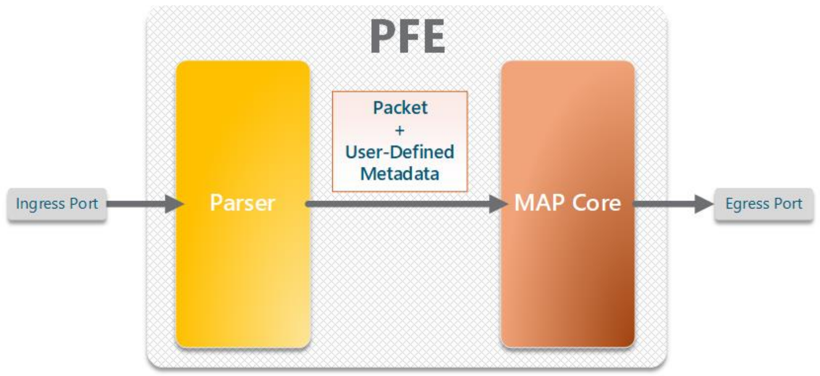
1. A Packet ingresses on a port, and is directed to the PFE along with its port type, which in turn, determines the Parser and MAP Instructions to be executed.
2. The Packet header is parsed by the Parser program. The Parser can parse L2/L3/L4 headers belonging to any standard, emerging, or proprietary network protocol.

   The architecture supports a flexible division of tasks between the Parser and the MAP due to the following properties:

   - The MAP has access to the entire packet header.
   - The Parser is capable of conveying user-defined metadata to a MAP core.
3. Packet is processed by the MAP program and forwarded to an egress port or discarded. The MAP program handles packet processing by applying various types of operations on the packet, as dictated by the application requirements, enabling protocol-independent processing. The order of operations in the MAP is flexible, allowing for a combination of sequential and parallel operations. In addition, the MAP allows for flexible editing of packet headers.

The packet handoff between the Parser and the MAP is flexible.

Figure 2: PFE Packet Processing Flow



3 Parser Programming Model

This section provides detailed descriptions and specifications for the X-Switch device Programmable Parser ISA instruction set.

3.1 Overview

The Parser is a programmable packet processing engine.

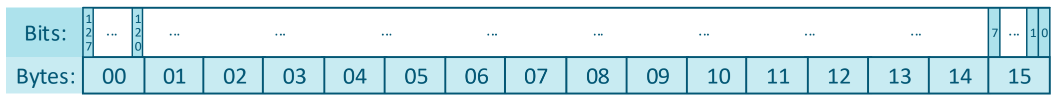
The main task of the Parser is to parse the incoming packet header according to the collocated Parser transition table, and to provide the information by setting the appropriate HDR.PRESENT and HDR.OFFSET0/1 metadata associated with the packet as it is assigned to a MAP thread for further processing. Refer to STC, STCI, STH instructions for details.

The Parser can parse header sizes of up to 256B. Attempting to move the cursor beyond the header size, results in a header-violation error.

3.2 Register Structure

The Parser supports 4 general purpose registers: `R0` - `R3`.
Each register contains 128b, as shown in Figure 3.

Figure 3: Register Addressing Modes - Bit/Byte



The Parser program supports Big-Endian architecture:

- MSbit is #127, MS byte is #0.
- LSbit is #0, LS byte is #15.

In certain operations, where the destination register is not required, a null register (`RN`) is supported.

3.3 Parser Program Termination

The Parser ends packet processing by using a HALT instruction. At this point, packet processing continues by a MAP thread.

If the Parser program terminates abnormally, the status is indicated in specific Parser status flags.

3.4 Parser Cursor

The Parser maintains a cursor on the incoming packet data buffer.
Many of the instructions support operations based on the cursor location.

3.5 Parser Transition Table

The Parser transition table is the main data structure which controls the Parser operation, along with the Parser microcode.
The table includes ***transition rules***, where each rule includes:

- Transition key: parse state, protocol value
- Next Parser state

Whenever the program performs a `NXTP` instruction with a specific protocol key, the key {current-state, protocol-key} is searched in the table.

- If a match is found, the next state is determined.
- If no match is found, the program continues according to `JumpMode`, see Section 3.5.1**.**

3.5.1 JumpMode: Transition to Next State

When an `NXTP` instruction is performed, the Parser calculates the next state, as described in Section 3.5, as a background operation. A subsequent branch instruction is required in order to move to the next state entry point found. The branch instruction can be either explicit, or, in conjunction with another instruction (STH, STC, STCI, STCH, STHC) and is controlled by a `JumpMode` operand. This operand determines whether to perform a branch to next state, and how to proceed in case of a **no match**:

- **000**: No jump is performed.

  Relevant for STH, STC, STCI, STCH, STHC instructions; N/A for a branch instruction.
- Other values indicate how to act upon next-protocol failure:
  - **001**: Continue to next instruction.
  - **010**: Jump to explicit address.
    - Relevant only in branch instructions;
      the branch address is specified in the instruction.
    - No change in state.
  - **011**: Transition to a new state.

    Relevant only in branch instructions;
    the transition rule number is specified in the instruction.
  - **100**: Trap. Jump to a pre-configured trap address for handling errors.

    Relevant for STH, STC, STCI, STCH, STHC instructions; N/A for branch instructions (since an explicit address can be specified by: `JumpMode 010`).

3.6 Protocol Seek Accelerator (PSEEK)

The PSEEK instruction is a Parser accelerator that enables scanning a series of protocol headers, and ***fast-forwarding*** the Parser cursor up to the next protocol header of interest.

A configuration table allows the user to define the set of protocols to skip over.

Each entry includes the information required to skip the designated header:

- Header /Protocol ID offset and value.
- Next Header offset and size.

A header violation error is incurred when PSEEK attempts to read headers longer than 256B and the seek is aborted.

3.7 IPv4 Verify Checksum Accelerator

The Parser supports a checksum-verify accelerator that can be activated either via a dedicated instruction, or in conjunction with other instructions.

The checksum is calculated as the cursor moves from the designated starting point in the packet to the end point. No action is taken when the result is correct (=0).

If however the result is a bad checksum, a trap is triggered, interrupting the program flow and causing it to continue at a pre-configured PC address for handling the trap.

3.8 Standard Metadata (SMD)

The Parser prepares the ***Standard Metadata (SMD)*** for the MAP. The SMD includes various information, including parser errors.

3.9 Packet Reparse

The Parser supports receiving a packet header from the MAP core for reparsing (see REPARSE instruction in MAP-ISA programmers guide).

The packet header is received with additional information:

- **Cursor offset:** relative to **FOF** (Frame Offset), from which to start the parsing.
- **Cookie:** An 8b value conveyed to the Parser program, and is stored in Parser register **R3[127:120]**.

The Parser program starts at a designated `REPARSE` entry point. During the re-parsing program, Parser results are stored in internal storage, and at the end of the program, the results must be copied to pre-designated MAP registers (using the `MOVMAP.HDR` instruction). The results include `HDR.PRESENT`, `HDR.OFFSET0/1`, and a copy of the SMD's first 4 bytes.

In case of abnormal program termination due to hardware failures (e.g., bad checksum, header size violation), the Parser program jumps to the Parser trap entry point, and status flags are used as with normal parsing programs, however those are not stored in the SMD but in the copy of the SMD's first 4 bytes, which can be copied to a MAP register.

The Parser program always returns to the MAP SYNC point, also when the Parser program ends with a `DROP` instruction. In such cases, the SYNC results with a failure.

The Parser supports special `HALT.RP` and `HALTDROP.RP` instructions:

- In normal parsing scenarios, the program halts or drops the packet and terminates.
- In reparse scenario, the program can specify a jump address to perform further processing before terminating, e.g., move the parsing results to MAP registers.

3.10 Instruction Syntax Conventions

Table 1: Instruction Syntax Conventions

Acronym	Derivatives/Examples	Description
`Rs`	`RsS, RsE`	Source register (Derivatives: a range of {Start register, end register})
`Rd`	`RdS, RdE`	Destination register
`Z`		Zero flag
`N`		Negative flag
`[Opt]`	`MOVI.CD, ETX.SCSM`	`Opt` is an optional instruction extension: `[.H]`, `[.CD]`, `[.PR]`, `[.SCSM]`, `[.ECSM]`
`[.H]`		Halt the program after the operation.
`[.CD]`		Clear destination register before the operation.
`[.PR]`	EXT.PR	Add present-bit.
`[.SCSM]`		Start checksum calculation from current cursor position.
`[.ECSM]`		End checksum calculation at current cursor position.

3.11 Instruction Set

This section describes each Parser ISA instruction. Table 2: **Instruction Summary** is followed by detailed information for each instruction or instruction set in the following sub-sections.

Table 2: Instruction Summary

Category	Instruction	Operation	Link
Parse Graph Traversal	NXTP	Parse next protocol	Section 3.12.1
Protocol Seek Traversal	PSEEK	Fast-forward over protocol headers	Section 3.12.2
Load/Store	EXT	Extract data from a packet	Section 3.12.3
	EXTMAP	Extract data from a packet into a MAP register	Section 3.12.4
	MOVMAP	Move data from a Parser register into a MAP register	Section 3.12.5
	CNCT	Concatenate data	Section 3.12.6
	STH	Set Header fields	Section 3.12.7
	STC	Set cursor	Section 3.12.8
	STCH	Set cursor and Header fields	Section 3.12.9
	ST	Store data in Struct	Section 3.12.10
Bit/Byte Manipulation	MOV	Move data	Section 3.12.11
	MOVL, MOVR	Move data left/right	Section 3.12.12
Arithmetic and Logical	ADD	Add	Section 3.12.13
	SUB	Subtract	Section 3.12.14
	AND	Bitwise AND	Section 3.12.15
	OR	Bitwise OR	Section 3.12.16
	CMP	Compare	Section 3.12.17
Jump/Branch	BR	Branch by condition	Section 3.12.18
Misc	HALT	Terminate program	Section 3.12.19
	NOP	No operation	Section 3.12.20

3.12 Instruction Details

3.12.1 NXTP

Function: Calculate *next-protocol* entry address.

That is, it calculates the new PC that will serve as an entry point for the next parser state, using the contents of the specified register (up to 24 bits) as a part of the key in the transition table.

Syntax

```
NXTP Rs, Offset, Size
```

Instruction Format

NXTP: NXTP SourceReg, SourceOffsetBits, SizeBits

Instruction Operation: NXTP (key = SourceReg[i-1:j])

Calculates next-protocol by extracted key; key can be up to 24 bits.

The operation is asynchronous, and the result is the PC entry point for next protocol. The program branches to the next entry point upon a subsequent jump instruction (see Section 3.5.1, JumpMode: Transition to Next State for further details).

• **j:** SourceOffsetBits:	Range: 0 - 127.
• **i:** SourceOffsetBits + SizeBits:	Range: 1 - 24.

Timing Characteristics: Asynchronous operation.

3.12.2 PSEEK, PSEEKNXTP

Function: Scan over protocol headers up to the next protocol of interest, based on the pre-configured PSEEK table.

Syntax

PSEEK: `PSEEK[.CD] Rd, destOffset, Rs, srcOffset, Size, ClassId`

PSEEKNXTP: `PSEEKNXTP[.CD] Rd, destOffset, Rs, srcOffset, Size, ClassId`

Instruction Format

PSEEK: PSEEK DestReg, DestOffsetBits, SrcReg, SourceOffsetBits, SizeBits, ClassID

PSEEKNXTP: PSEEKNXTP DestReg, DestOffsetBits, SrcReg, SourceOffsetBits, SizeBits, ClassID

Instruction Operation:

- **PSEEK:** Scan over protocol headers to be skipped;
  DestReg[i:j] = next protocol to process.
- **PSEEKNXTP:** Scan over protocol headers to be skipped;
  DestReg[i:j] = next protocol to process.
  Perform **NXTP** (calculate the next protocol to process with the key in DestReg).
  - **DestReg** can be RN (null register) if there is no need to get the next protocol value.
- The next protocol field length and size are taken from the last entry found in the PSEEK table entry.
- **DestOffsetBits:** Range: 0 - 127.

The cursor is moved forward over protocol headers specified in the PSEEK table (see Protocol Seek Accelerator (PSEEK)), up to a protocol header that is not specified in the table. This protocol ID is placed in DestReg, and can be used by a subsequent NXTP instruction.

The scan starts with a search of the protocol field specified in SrcReg, at offset SourceOffsetBits and size SizeBits.

- **Class-ID:** Value used for partitioning the table into logically separate tables for different needs.
- **DestOffsetBits:** Range: 0-127.
- **SourceOffsetBits:** Range: 0-127.
- **SizeBits:** Range: 1-16.
- **ClassID:** Range: 0-3.

> - If a protocol length field calculation results with 0, or reaches beyond the first packet header size, the PSEEK command terminates its operation, and a ***PSEEK_ERROR*** status flag is set;
>   the program continues at the special Parser Trap address.
> - If there is no need for the PSEEK result, DestReg can be the Null register (RN).

Timing Characteristics: Asynchronous operation.

3.12.3 EXT, EXTNXTP

Function:

- **EXT:** Extract data from packet to destination register.
- **EXTNXTP:** Extract data from packet and calculate *next-protocol* entry address.

Syntax

EXT:	`EXT[.CD][.SCSM][.PR] Rd, DestOffset, SourceOffset, Size`
EXTNXTP:	`EXTNXTP[.CD][.SCSM][.PR] Rd, SourceOffset, Size`

Instruction Format

EXT:	EXT[.CD][.SCSM][.PR] DestReg, DestOffsetBits, SourceOffsetBits, SizeBits
EXTNXTP:	EXTNXTP[.CD][.SCSM][.PR] DestReg, SourceOffsetBits, SizeBits

Instruction Operation:

- **EXT:** DestReg[i-1:j] = (Packet[l:k-1])

  Extract data from a packet to a register.

❍ **j:** DestOffsetBits	Range: 0 - 127.
❍ **i:** DestOffsetBits + Sizebits	Range: 1 - 128.
❍ **l:** SourceOffsetBits	Offset from current cursor position; Range: 0 - 511
❍ **k:** SourceOffsetBits + SizeBits	Range: 1 - 128.

  ❍ **.PR:** When opted, **1** is appended as MSbit to the extracted field:

■ DestReg[i-1:j] =	Packet[l:k-1]
■ DestReg[i:i] =	1

  When `EXT.PR` is opted, size must be less than 128.

- **EXTNXTP:** DestReg[i-1:0] = (Packet[l:k-1]); NXTP (key = Packet[l:k-1])

  Extract up to 24b key from packet to a register and calculate next-protocol by extracted key.

  ❍ **DestReg** can be `RN` (null register) if there is no need to get the next protocol value.

❍ **i:** Sizebits:	Range: 1 to 24.
❍ **l:** SourceOffsetBits	Offset from current cursor position; Range: 0 - 511.
❍ **k:** SourceOffsetBits + SizeBits:	Range: 1 to 24.

  ❍ **.PR:** When opted, **1** is appended as MSbit to the extracted field:

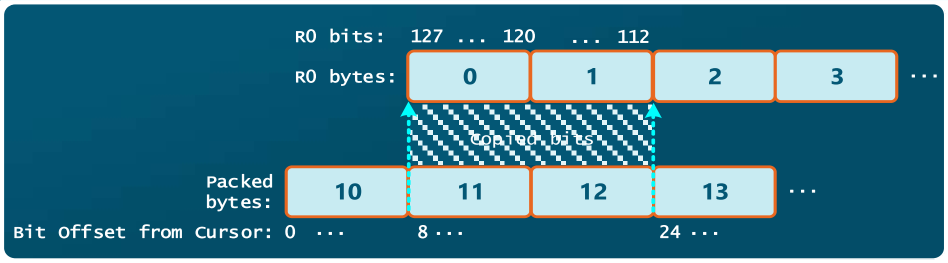
  ■ DestReg[i-1:0] = (Packet[l:k-1])
  ■ DestReg[i:i] = 1

  When `EXTNXTP.PR` is opted, the additional present-bit is not included in the key search.

Up to 128 bits can be extracted, however only the first 24 bits (LSBits) can be used as the key for next-protocol calculation.

Example: `EXT R0, 112, 8, 16`

• DestReg =	R0
• DestOffsetBits =	112
• Cursor position =	10
• SrcOffsetBits=	8
• SizeBits =	16


R0 bits: 127 ... 120 ... 112
R0 bytes:
0
1
2
3
...
Copied bits
Packed bytes:
10
11
12
13
...
Bit Offset from Cursor: 0 ... 8 ... 24 ...

- **.SCSM:** When specified;
  The Parser HW starts calculating checksum
  from current cursor position.
  The programmer must specify a matching **.ECSM** option with a set-cursor instruction at the end of the check-summed header.

Timing Characteristics:

- **EXT**: Single Cycle
- **EXTNXTP**: Asynchronous operation.

3.12.4 EXTMAP

Function: Extract data from packet to a MAP register.

Syntax

EXTMAP: `EXTMAP[.PR] MAPRd, DestOffset, PacketOffset, Size`

Instruction Format

EXTMAP: EXTMAP[.PR] MAPReg, DestOffsetBits, PacketOffsetBits, SizeBits

Instruction Operation: MAPReg[i-1:j] = (Packet[l:k-1])

Extract data from packet to a MAP register.

• **MAPReg:**	MAP register number; Range: 0 - 13.
• **j:** DestOffsetBits:	Range: 0 - 127.
• **i:** DestOffsetBits + Sizebits:	Range: 1 - 128.
• **l:** PacketOffsetBits:	Offset from current cursor position; Range: 0 - 511.
• **k:** PacketOffsetBits + SizeBits:	Range: 1 - 128.

- **.PR:** When opted, **1** is appended as MSbit to the extracted field:
  - MAPReg[i-1:j] = Packet[l:k-1]
  - MAPReg[i:i] = 1

Timing Characteristics: Single Cycle

3.12.5 MOVMAP

Function: Move data from a Parser register or Parser result register to a MAP register.

Syntax

MOVMAP: `MOVMAP[.HDR] MAPRd, DestOffset, Rs, SrcOffset, Size`

Instruction Format

MOVMAP: MOVMAP[.HDR] MAPReg, DestOffsetBits, SrcReg, SrcOffsetBits, SizeBits

Instruction Operation: MAPReg[di-1:dj] = SrcReg[si-1:sj]

Move data from a Parser register or a Parser-result register or SMD data (see Section 3.9, Packet Reparse) to a MAP register.

- **MAPReg:** MAP register number;
  Range: 0 - 13.
- **SrcReg:**
  - When **.HDR** is ***not*** opted: A Parser register (0 - 3).
  - When **.HDR** ***is*** opted: A Parser result register or SMD data:

    0: HDR.PRESENT result register.

    1: HDR.OFFSET0 result register.

    2: HDR.OFFSET1 result register.

    3: The SMD first 4 bytes (or less, depending on SrcOffsetBits and SizeBits) are written to the **MAPReg**, however only the following fields are valid:

    - Last Parser state.
    - HW failure status bits.
    - SAF mode.
    - IDMA mode.
- **dj:** DestOffsetBits: Range: 0 - 127.
- **di:** DestOffsetBits + Sizebits: Range: 1 - 128.
- **sj:** SrcOffsetBits: Range:
  - For Parser/Parser-result register: 0 - 127;
  - For SMD: 0 - 31.
- **si:** SrcOffsetBits + Sizebits: Range:
  - For Parser/Parser result register: 1 - 128;
  - For SMD: 1 - 32

Timing Characteristics: Single Cycle

3.12.6 CNCTBY, CNCTBI

Function: Concatenate data.

Syntax

CNCTBY: `CNCTBY[.CD] Rd, DestOffset, Rs1, Src1Offset, Src1Size, Rs2, Src2Offset, Src2Size`

CNCTBI: `CNCTBI[.CD] Rd, DestOffset, Rs1, Src1Offset, Src1Size, Rs2, Src2Offset, Src2Size`

Instruction Format

CNCTBY: CNCTBY[.CD] DestReg, DestOffsetBytes, SrcReg1, Src1OffsetBytes, Src1SizeBytes, SrcReg2, Src2OffsetBytes, Src2SizeBytes

CNCTBI: CNCTBI[.CD] DestReg, DestOffsetBits, SrcReg1, Src1OffsetBits, Src1SizeBits, SrcReg2, Src2OffsetBits, Src2SizeBits

Instruction Operation: Concatenate data from two source registers to destination register.

CNCTBY: Offsets and sizes are in 8b granularity.

CNCTBI: Offsets and sizes are in 1 bit granularity.

- Range for all Offsets: 0 - 15.
- Range for all Sizes: 1 - 16.
- All offsets relate to ***bit offsets*** in a register, according to bit ordering convention.

Timing Characteristics: Single Cycle

3.12.7 STH

Function: Set Header present and offset fields, and optionally jump to next protocol entry point.

Syntax

STH: `STH[.SCSM|.ECSM][.H] HdrID, HdrOffsetID, JumpMode`

Instruction Format

STH: STH[.SCSM|.ECSM][.H] HeaderPresentID, HeaderOffsetID , JumpMode

Instruction Operation:

- HDR.PRESENT[HeaderPresentID] = 1
- HDR.OFFSET[HeaderOffsetID] = Cursor position
- **JumpMode:** See Section 3.5.1, JumpMode: Transition to Next State for details.
- **.SCSM:** When specified, the Parser HW starts calculating checksum from current cursor position. The programmer must specify a matching **.ECSM** option with a set-cursor instruction at the end of the check-summed header.
- **.ECSM:** When specified, the Parser HW finalizes checksum calculation at one byte before the cursor position. If checksum is bad, the program jumps to a pre-configured PC address for handling abnormal termination.
- **.H:** When specified, halt (terminate the current packet parsing) after the operation.

> In reparse flow **.H** is ignored, and the program continues execution in next instruction.

Timing Characteristics: Single Cycle

3.12.8 STC, STCI

Function: Set cursor position, and optionally jump to next protocol entry point.

Syntax

STC: `STC[.SCSM|.ECSM] Rs, SrcOffset, SrcSize, SizeShift, AdditionalIncr, JumpMode`

STCI: `STCI[.SCSM|.ECSM] IncrValue, JumpMode`

Instruction Format

STC: STC[.SCSM|.ECSM] SrcReg, SrcOffsetBits, SrcSizeBits, SrcShift, AdditionalIncr, JumpMode

STCI: STCI[.SCSM|.ECSM] IncrValue, JumpMode

Instruction Operation:

- **STC:** Cursor += ((SrcReg[i:j] + AdditionalIncr) << SrcShift) ; Jump according to JumpMode

  The cursor is a byte location in the packet. It is incremented by a value extracted from the source register added with AdditionalIncr, then multiplied by 2^SrcShift.

❍ **j:** SrcOffsetBits:	Range: 0-127.
❍ **i:** SrcOffsetBits + SrcSizeBits:	Range: 1-8.
❍ **SrcShift:**	Range: 0 - 7.
❍ **AdditionalIncr:**	Range: 0 - 3.

- **STCI:** Cursor += IncrValue Jump according to JumpMode

  The cursor is incremented by IncrValue.

❍ **IncrValue:**	Range: 1-256.

- **JumpMode:** See Section 3.5.1, JumpMode: Transition to Next State for details.
- **.SCSM:** When specified, the Parser HW starts calculating checksum from current cursor position. The programmer must specify a matching **.ECSM** option with a set-cursor instruction at the end of the check-summed header.
- **.ECSM:** When specified, the Parser HW finalizes checksum calculation at one byte before cursor position. If checksum is bad, the program jumps to a pre-configured PC address for handling abnormal termination.

Timing Characteristics:

For **JumpMode**=0: Single Cycle;

Otherwise, Asynchronous operation.

3.12.9 STCH, STHC

Function: Set cursor position, set Header present and offset fields, and optionally jump to next protocol entry point.

Syntax

STCH:	`STCH[.SCSM|.ECSM][.H] IncrValue, HdrID, HdrOffsetID, JumpMode`
STHC:	`STHC[.SCSM|.ECSM]     IncrValue, HdrID, HdrOffsetID, JumpMode`

Instruction Format

STCH: STCH[.SCSM|.ECSM][.H] Offset, HeaderPresentID, HeaderOffsetID, JumpMode

Instruction Operation: Compound operation of **STCI** followed by **STH**:

The operation performs the following actions in this order:

1. Cursor location is incremented by Offset (IncrValue).
2. Header present-bit and offset field are updated.
3. Jump according to **JumpMode** (see description in [STH]).

.H: When specified, halt (terminate the current packet parsing) after the operation.

> In reparse flow **.H** is ignored, and the program continues execution in next instruction.

Instruction Format

STHC: STHC[.SCSM|.ECSM] Offset, HeaderPresentID, HeaderOffsetID, JumpMode

Instruction Operation: Compound operation of **STH** followed by **STCI**:

The operation performs the following actions in this order:

1. Header present-bit and offset field are updated according to current cursor position.
2. The cursor location is incremented by IncrValue.
3. Jump according to **JumpMode,** see Section 3.5.1, JumpMode: Transition to Next State for details.).

- **.SCSM:** When specified, the Parser HW starts calculating checksum from the new cursor position, following the current instruction. The programmer must specify a matching **.ECSM** option with a set-cursor instruction at the end of the check-summed header.
- **.ECSM:** When specified, the Parser HW finalizes checksum calculation at one byte before cursor position. If checksum is bad, the program jumps to a pre-configured PC address for handling abnormal termination.

Timing Characteristics: For **JumpMode**=0: Single Cycle;

Otherwise, Asynchronous operation.

3.12.10 ST, STI

Function: Store data in Struct-0.

Syntax

ST: `ST[.H] Rs, SrcOffsetBits,  StructOffsetBits, SizeBits`

STI: `STI ImmediateValue, StructOffsetBits, SizeBits`

Instruction Format

ST: ST[.H] SrcReg, SrcOffsetBits, StructOffsetBits, SizeBits

STI: STI ImmediateValue, StructOffsetBits, SizeBits

Instruction Operation: ST: Struct[l:k] = SrcReg[i:j]

- **l:** StructOffsetBits: Range: ***0-5 and 32-27***; Where Struct bits are numbered [0,..., 127].
- **k:** StructOffsetBits + SizeBits: ***with the exception of HW controlled bits 6-31***).
- **j:** SrcOffsetBits: Range: 0-127; Where SrcReg bits are numbered [127,...,0].
- **i:** SrcOffsetBits+ SizeBits
- **SizeBits:** Range: 1 - 128.
- **.H:** When specified, halt (terminate the current packet parsing) after the operation.

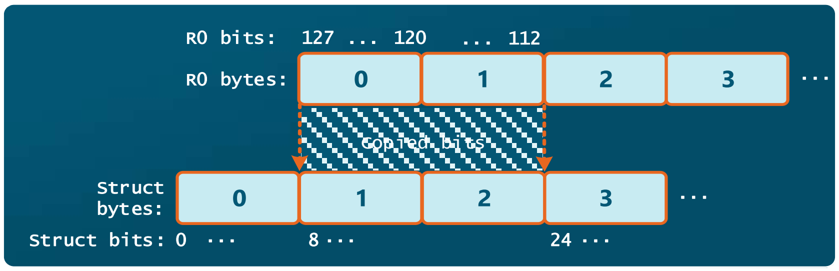
> In reparse flow **.H** is ignored, and the program continues execution in next instruction.

Instruction Operation: STI: Struct[l:k] = ImmediateValue

- **l:** StructOffsetBits Range: 0-127; Where Struct bits are numbered [0,..., 127].
- **k:** StructOffsetBits + SizeBits
- **SizeBits:** Range: 1-16.
    - **ImmediateValue** Max: 16b value.

Example:

`SrcReg = R0, SrcOffset = 112, StructOffset = 8, Size = 16`



Timing Characteristics: Single Cycle

3.12.11 MOV, MOVI

Function: Move data to register.

Syntax

MOV:	`MOV[.CD]   Rd, DestOffsetBits, Rs, SrcOffsetBits, SizeBits`
MOVI:	`MOVI[.CD]  Rd, DestOffsetBytes, ImmediateValue, SizeBits`

Instruction Format

MOV: MOV[.CD] DestReg, DestOffsetBits, SourceReg, SrcOffsetBits, SizeBits

MOVI: MOVI[.CD] DestReg, DestOffsetBytes, ImmediateValue, SizeBits

Instruction Operation: Copy value from source register or immediate-value to destination register:

- **MOV:** DestReg[i:j] = SourceReg[k:l]
  - **j:** DestOffsetBits — Range: 0-127.
  - **i:** DestOffsetBits + SizeBits — Range: 1-128.
  - **l:** SrcOffsetBits — Range: 0-127.
  - **k:** SrcOffsetBits + SizeBits — Range: 1-128.
- **MOVI:** DestReg[i:j] = ImmediateValue[k-1:0]
  - **j:** (DestOffsetBytes * 8)
  - **i:** (DestOffsetBytes * 8) + SizeBits
  - **k:** SizeBits
  - DestOffsetBytes: — Range: 0-15.
  - SizeBits: — Range: 1-16.
    - ImmediateValue: — Max: 16b value.

Timing Characteristics: Single Cycle

3.12.12 MOVL, MOVLI, MOVLII, MOVR, MOVRI, MOVRII

Function: Move data to left/right.

Syntax

MOVL:	`MOVL[.CD]`	`Rd, Rs1, offsBits1, SizeBits1, Rs2, OffsBits2, SizeBits2`
MOVLI:	`MOVLI[.CD]`	`Rd, Rs, offsBits, SizeBits, ImmVal`
MOVLII:	`MOVLII[.CD]`	`Rd, Rs, offsBits, SizeBits, ImmVal, ImmValSize`
MOVR:	`MOVR[.CD]`	`Rd, Rs1, offsBits1, SizeBits1, Rs2, OffsBits2, SizeBits2`
MOVRI:	`MOVRI[.CD]`	`Rd, Rs, offsBits, SizeBits, ImmVal`
MOVRII:	`MOVRII[.CD]`	`Rd, Rs,offsBits, SizeBits, ImmVal, ImmValSize`

Instruction Format

MOVL: MOVL[.CD] DestReg, SrcReg1, OffsBits1, SizeBits1, SrcReg2 OffsBits2, SizeBits2

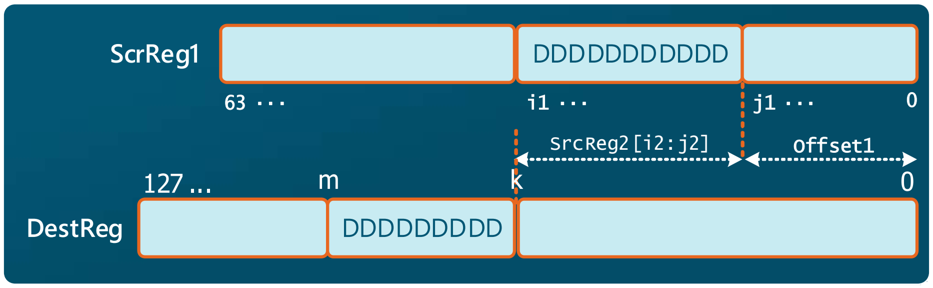
Instruction Operation: DestReg[m:k] = SrcReg1[i1:j1]

- **k = SrcReg2[i2:j2] + offsBits1**, bit offset in DestReg to which the source data is moved.
- **m = k + SizeBits1 - 1**, the size of the source data;

  Where:
  - **m** <= 63, i.e. in case of overflow, the source data MSbits are truncated.
  - **m - k** <= SizeBits1

• **j1, j2:** OffsBits1/OffsBits2	Range: 0 - 63.
• **i1:** OffsBits1 + SizeBits1	Range: 1 - 32.
• **i2:** OffsBits2 + SizeBits2	Range: 1 - 8.

Illustrated Operation:



Instruction Format

MOVLI: MOVLI[.CD] DestReg, SrcReg, OffsBits, SizeBits, ImmValue

Instruction Operation: DestReg[m:k] = SrcReg1[i1:j1]

- **k = ImmValue + offsBits**
- **m = k + SizeBits - 1**
  - **m - k <= SizeBits**

• **j:** OffsBits	Range: 0 - 127.
• **i:** OffsBits + SizeBits	Range: 1 - 32.
• **ImmValue:**	Range: 0 - 127.

Instruction Format

MOVLII: MOVLII[.CD] DestReg, SrcReg, offsBits, SizeBits, ImmValue, ImmValueSize

Instruction Operation: DestReg[m-1:k] = ImmValue[n-1:0]

- **k:** SrcReg[i:j]
- **m:** k + ImmValueSize
  - **m - k <= ImmValueSize**

• **j:** offsBits	Range: 0 - 127.
• **i:** offsBits + SizeBits	Range: 1 - 7.
• **ImmValue**	Range: 0 - 127.
• **ImmValueSize:**	Range: 1 - 7. Units: Bits.
• **n:** ImmValueSize	

Instruction Format

MOVR: MOVR[.CD] DestReg, SrcReg1, OffsBits1, SizeBits1, SrcReg2 OffsBits2, SizeBits2

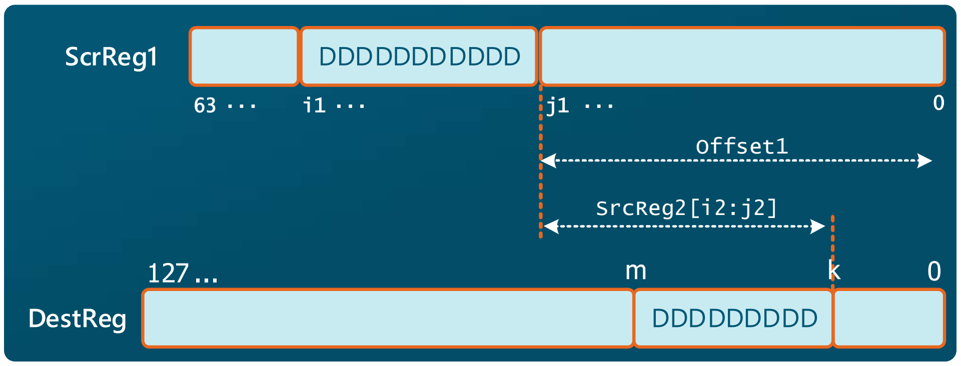
Instruction Operation: DestReg[m:k] = SrcReg1[i1:j1]

- **k = OffsBits1 - SrcReg2[i2:j2]**
- **m = OffsBits1 - SrcReg2[i2:j2] + SizeBits1 - 1**

  Where:
  - **k** >=0, i.e. in case of underflow, the source data LSbits are truncated
  - **m - k** <= SizeBits1

j1, j2: OffsBits1/OffsBits2	Range: 0 - 63.
i1: OffsBits1 + SizeBits1	Range: 1 - 32.
i2: OffsBits2 + SizeBits2	Range: 1 - 8.

Illustrated operation:



Instruction Format

MOVRI: MOVRI[.CD] DestReg, SourceReg, OffsBits, SizeBits, ImmValue

Instruction Operation: DestReg[m:k] = SrcReg1[i1:j1]

- **k:** OffsBits - ImmValue
- **m:** k + SizeBits - 1
  - **m - k <= SizeBits**

j: offsBits	Range: 0 - 127.
i: offsBits + SizeBits	Range: 1 - 32.
ImmValue	Range: 0 - 127.

Instruction Format

MOVRII: MOVRII[.CD] DestReg, SourceReg, OffsBits, SizeBits, ImmValue, ImmValueSize

Instruction Operation: DestReg[m-1:0] = ImmValue[n-1:k]

- **k:** SrcReg[i:j]
- **n:** ImmValueSize
- **m:** ImmValueSize - SrcReg[i:j]

j: OffsBits	Range: 0 - 127.
i: OffsBits + SizeBits	Range: 1 - 7.
ImmValue	Range: 0 - 127.
ImmValueSize:	Range: 1 - 7. Units: Bits.

Timing Characteristics: Single Cycle

3.12.13 ADD, ADDI

Function: Addition

Syntax

ADD: `ADD[.CD]  Rd, DestOffsetBits, Rs1, Src1OffsetBits, Rs2, Src2OffsetBits, SizeBits`

ADDI: `ADDI[.CD]  Rd, Rs, ImmediateValue, SizeBits`

Instruction Format

ADD: ADD[.CD] DestReg, DestOffsetBits, Src1Reg, Src1OffsetBits, Src2Reg, Src2OffsetBits, SizeBits

Instruction Operation: DestReg[k:l] = Src1Reg[i1:j1] + Src2Reg[i2:j2]

Supports 16b integer addition: total of offset + size must be ≤ 16b.

Operands are treated as unsigned integers.

• **l:** DestOffsetBits	Range: 0-15.
• **k:** DestOffsetBits + SizeBits	Range: 1-16.
• **j:** SrcOffsetBits	Range: 0-15.
• **i:** SrcOffsetBits + SizeBits	Range: 1-16.

Instruction Format

ADDI: ADDI[.CD] DestReg, SrcReg, ImmediateValue, SizeBits

Instruction Operation: DestReg[i:0] = SrcReg[i-1:0] + ImmediateValue

Operands are treated as unsigned integer.

• **ImmediateValue:**	15b value
• **i:** SizeBits	Range: 1-15.

Flags Affected:

Z: Set if the result is zero .

Only result bits are considered, regardless of carry/overflow status.

Timing Characteristics: Single cycle

3.12.14 SUB, SUBI, SUBII

Function: Subtraction

Syntax

SUB: `SUB[.CD]  Rd, DestOffsetBits, Rs1, Src1OffsetBits, Rs2, Src2OffsetBits, SizeBits`

SUBI: `SUBI[.CD]  Rd, Rs, ImmediateValue, SizeBits`

SUBII: `SUBII[.CD]  Rd, ImmediateValue, Rs, SizeBits`

Instruction Format

SUB: SUB[.CD] DestReg, DestOffsetBits, Src1Reg, Src1OffsetBits, Src2Reg, Src2OffsetBits, SizeBits

Instruction Operation: DestReg[k:l] = Src1Reg[i1:j1] - Src2Reg[i2:j2]

Supports 16b integer subtraction: total of offset + size is 16b.

Both operands are treated as unsigned integers, however the result may be negative (signed).

• **l:** DestOffsetBits:	Range: 0-15.
• **k:** DestOffsetBits + SizeBits:	Range: 1-16.
• **j:** SrcOffsetBits:	Range: 0-15.
• **i:** SrcOffsetBits + SizeBits:	Range: 1-16.

Instruction Format

SUBI: SUBI[.CD] DestReg, SrcReg, ImmediateValue, SizeBits

Instruction Operation: DestReg[i-1:0] = SrcReg[i-1:0] - ImmediateValue

SUBII: SUBII[.CD] DestReg, ImmediateValue, SrcReg, SizeBits

Instruction Operation: DestReg[i-1:0] = ImmediateValue - SrcReg[i-1:0]

- **ImmediateValue:** 15b value

• **i:** SizeBits:	Range: 1-15.

Flags Affected:

Z: Set if result is zero.

N: Set if result is negative.

Timing Characteristics: Single Cycle

3.12.15 AND, ANDI

Function: Bitwise AND

Syntax

AND: `AND[.CD]  Rd, DestOffsetBits, Rs1, Src1OffsetBits, Rs2, Src2OffsetBits, SizeBits`

ANDI: `ANDI[.CD]  Rd, Rs, ImmediateValue, SizeBits`

Instruction Format

AND: AND[.CD] DestReg, DestOffsetBits, Src1Reg, Src1OffsetBits, Src2Reg, Src2OffsetBits, SizeBits

Instruction Operation: DestReg[k:l] = Src1Reg[i1:j1] & Src2Reg[i2:j2]

• **l:** DestOffsetBits:	Range: 0-15.
• **k:** DestOffsetBits + SizeBits:	Range: 1-16.
• **j:** SrcOffsetBits:	Range: 0-15.
• **i:** SrcOffsetBits + SizeBits:	Range: 1-16.

Instruction Format

ANDI: ANDI[.CD] DestReg, SrcReg, ImmediateValue, SizeBits

Instruction Operation: DestReg[i-1:0] = SrcReg[i-1:0] & ImmediateValue

• **ImmediateValue:**	15b value.
• **i:** SizeBits	Range: 1-15.

Flags Affected:

Z: Set if result is zero.

Timing Characteristics: Single Cycle

3.12.16 OR, ORI

Function: Bitwise OR

Syntax

OR: `OR[.CD]  Rd, DestOffsetBits, Rs1, Src1OffsetBits, Rs2, Src2OffsetBits, SizeBits`

ORI: `ORI[.CD]  Rd, Rs, ImmediateValue, SizeBits`

Instruction Format

OR: OR[.CD] DestReg, DestOffsetBits, Src1Reg, Src1OffsetBits, Src2Reg, Src2OffsetBits, SizeBits

Instruction Operation: DestReg[k:l] = Src1Reg[i1:j1] | Src2Reg[i2:j2]

• **l:** DestOffsetBits	Range: 0-15.
• **k:** DestOffsetBits + SizeBits	Range: 1-16.
• **j:** SrcOffsetBits	Range: 0-15.
• **i:** SrcOffsetBits + SizeBits	Range: 1-16.

Instruction Format

ORI: ORI[.CD] DestReg, SrcReg, ImmediateValue, SizeBits

Instruction Operation: DestReg[i-1:0] = SrcReg[i-1:0] | ImmediateValue

• **ImmediateValue**	15b value.
• **i:** SizeBits	Range: 1-15.

Flags Affected:

Z: Set if result is zero.

Timing Characteristics: Single Cycle

3.12.17 CMP, CMPIBY, CMPIBI

Function: Compare

Syntax

CMP:	`CMP  Rs1, Source1OffsetBits, Rs2, Source2OffsetBits, SizeBits`
CMPIBY:	`CMPIBY  Rs, SourceOffsetBytes, ImmediateValue, SizeBits`
CMPIBI:	`CMPIBI  Rs, SourceOffsetBits, ImmediateValue, SizeBits`

Instruction Format

CMP: CMP Source1, Source1OffsetBits, Source2, Source2OffsetBits, SizeBits

Instruction Operation: Result = Source1 [i1:j1] - Source2 [i2:j2]

- **j:** StartOffset — Range: 0-127. Units: Bits.
- **i:** StartOffset + Size — Range: 1-32. Units: Bits.

Flags Affected:

- **Z** flag = (Result == 0)
- **N** flag = (Result < 0)

Instruction Format

CMPIBY: CMPIBY SourceReg, SourceOffsetBytes, ImmediateValue, SizeBits

Instruction Operation: Result = (SourceReg [i-1:j] - ImmediateValue)

Compare with source-offset specified in 8b granularity.

- **SizeBits** — Range: 1 - 16.
- **ImmediateValue:** — 16b value.
- **SourceOffsetBytes:** — Range: 0-15: Bit offset specified in 8b granularity.
- **j:** (SourceOffsetBytes * 8)
- **i:** (SourceOffsetBytes * 8) + SizeBits

Instruction Format

CMPIBI: CMPIBI SourceReg, SourceOffsetBits, ImmediateValue, SizeBits

Instruction Operation: Result = (SourceReg [i-1:j] - ImmediateValue)

Compare with source-offset specified in bits.

• **SizeBits:**	Range: 1 - 16.
• **ImmediateValue:**	16b value.
• **SourceOffsetBits:**	Range: 0-15. Bit offset specified in 1b granularity.
• **j:** StartOffset	
• **i:** StartOffset + Size	Range: 1-16. **Size** units: Bits.

Flags Affected:

- **Z** flag = (Result == 0)
- **N** flag = (Result < 0)

Timing Characteristics: Single Cycle

3.12.18 BR, BRNS, BRBTST, BRNXTP, BRBTSTNXTP

Function: Branch with/without condition.

Syntax

BRcc: `BR[condition-code] PC`

BRNScc: `BRNS[condition-code] TransitionRule`

BRBTSTcc: `BRBTST[condition-code] Rs, SrcOffsetBits, PC`

BRNXTPcc: `BRNXTP[condition-code] JumpMode [, PC | TransitionRule]`

BRBTSTNXTPcc:`BRBTSTNXTP[condition-code] Rs, SrcOffsetBits, JumpMode [, PC | TransitionRule]`

Instruction Format

BRcc: BR<cc> Address

BRNScc: BRNS<cc> TransitionRule

BRBTSTcc: BRBTST<cc> SourceReg, SrcOffsetBits, Address

BRNXTPcc: BRNXTP<cc> JumpMode [, Address | TransitionRule]

BRBTSTNScc: BRBTSTNS<cc> SourceReg, SrcOffsetBits, TransitionRule

BRBTSTNXTPcc: BRBTSTNXTP<cc> SourceReg, SrcOffsetBits, JumpMode [, Address | TransitionRule]

Instruction Operation:

- **BRcc:** Branch to Address if condition is met.
- **BRNScc:** Branch to next state indicated by TransitionRule if condition is met.
- **BRBTSTcc:** Test bit value SourceReg[SrcOffsetBits], and branch to Address if condition is met.
  - **SrcOffsetBits** Range: 0-127.
- **BRNXTPcc:** Branch to next-protocol if condition is met, and in case of next-protocol failure, *jump to* according to `JumpMode`.
- **BRBTSTNXTPcc:** Test bit value SourceReg[SrcOffsetBits], branch to next-protocol if condition is met, and in case of next-protocol failure, *jump to* according to `JumpMode`.
- **BRBTSTNNScc:** Test bit value SourceReg[SrcOffsetBits], branch to next state indicated by TransitionRule if condition is met.
- **JumpMode:** See Section 3.5.1, JumpMode: Transition to Next State for details.
- Condition combinations listed here apply to **BR<cc> , BRNS<cc>,** and **BRNXTP<cc>**:
  - **BREQ:** Branch if Z = 1 (“branch if equal”).
  - **BRNEQ:** Branch if Z = 0 (“branch if not equal”).
  - **BRLT:** Branch if N = 1 (“branch if less than”).
  - **BRGT:** Branch if N = 0 and Z = 0 (“branch if higher than”).

- **BRGE:** Branch if N = 0 (“branch if greater or equal”).
- **BRLE:** Branch if N = 1 or Z = 1 (“branch if less or equal”).
- **BR:** Branch unconditionally.

- Condition combinations for **BRBTST**, **BRBTSTNS** and **BRBTSTNXTP**:
  - **BRBTSTCLR, BRBTSTNSCLR, BRBTSTNXTPCLR:** Branch if Z = 1 (“branch if clear”), i.e. tested bit = 0.
  - **BRBTSTSET, BRBTSTNSSET, BRBTSTNXTPSET:** Branch if Z = 0 (“branch if set”), i.e. tested bit = 1.

Timing Characteristics:

- Branch not taken: Single cycle
- Branch taken: Asynchronous operation

3.12.19 HALT, HALTDROP

Function: Terminate the current packet parsing, optionally drop the packet.
When reparsing, jump to a Parser PC.

Syntax

HALT: `HALT[.RP] [MAP-PC][, PARSER-PC]`

HALTDROP: `HALTDROP[.RP PARSER-PC]`

Instruction Format

HALT: HALT[.RP][MAP-PC][, PARSER-PC]

Instruction Operation:

HALT: Terminate the current packet parsing.
The packet is assigned to a MAP thread for further processing.

- **MAP-PC** Optional MAP entry point.
  When specified, the MAP thread is started at this entry point.
  If not specified, the default entry-point is selected (according to port-type and error status, see Section 3.3, Parser Program Termination.

  > In reparse scenario: **HALT** always returns to the MAP SYNC point,
  > and MAP-PC must not be specified.

HALT.RP: For normal entry point (not reparse), same as **HALT**.

For re-parsing, program jumps to PARSER-PC address for further processing.

- **PARSER-PC:** A valid Parser program address to jump to.

Table 3: Usage Examples

Instruction	Behavior in non-Reparse Scenario	Behavior in Reparse Scenario
HALT	Terminate the parsing program, MAP execution starts at designated entry-point according to port type.	Terminate the parsing program, MAP execution resumes at SYNC point.
HALT MAP-PC	Terminate the parsing program, MAP execution starts at MAP-PC.	Terminate the parsing program, MAP execution resumes at SYNC point.
HALT.RP PARSER-PC	Same as **HALT.**	Jump to PARSER-PC.
HALT.RP MAP-PC, PARSER-PC	Same as **HALT MAP-PC**.	Jump to PARSER-PC.

Instruction Format

HALTDROP: HALTDROP[.RP PARSER-PC]

Instruction Operation:

HALTDROP: Halt the current packet parsing.

- For non-reparse scenario, the packet is dropped.
- For reparse scenario, the packet is not dropped;
  MAP program resumes at the SYNC point, and SYNC result is failure.

HALTDROP.RP PARSER-PC: Halt the current packet parsing.

- For non-reparse scenario, the packet is dropped, PARSER-PC is ignored.
- For reparse scenario, the program jumps to PARSER-PC address.

> A multicast packet must not be dropped in the Parser.
> It must be conveyed to the MAP, where it can be dropped.

Timing Characteristics: Single Cycle

3.12.20 NOP

Function: No operation

Syntax

```
NOP
```

Instruction Format NOP

Instruction Operation: No operation.

Timing Characteristics: Single Cycle

4 MAP Programming Model

This section provides a detailed description and specifications for
the X-Switch device Match-and-Action Processor (MAP) ISA instruction set.

4.1 Overview

The MAP is a programmable packet processing engine.

4.2 Register Structure and Addressing Modes

The MAP supports 14 registers:

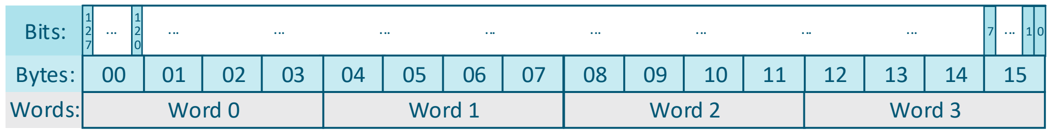
- **R0** to **R10**: 11 general purpose registers
- Three special, Packet Header (HDR) registers, are preloaded by the parser
    - **R11**: `HDR.PRESENT`
    - **R12**, **R13**: `HDR.OFFSET0/HDR.OFFSET1`
- **R14:** This is a virtual register, and is used in debug mode.
- **R15** and **RN** are aliases for a *null* register: They can be used as null destination registers in any operation where the result is not needed.

  For example, when the effect on ZNCV flags is needed (e.g., using `AND.F` instruction for testing a bit value).

4.3 Register Addressing Modes and Conventions

The registers contain 128 bits. Two addressing modes are supported as shown in Figure 4.

Figure 4: Register Addressing - Bit/Byte/Word



Each register is comprised of 4 words (Word 0 to Word 3). Each word is comprised of four bytes. Each byte is comprised of 8 bits, such that a full register is 128 bits.

The registers can be referenced either by the entire register of 128 bits by using `Ri`, or by its individual register words, such as `Ri.0`.

- Full Register (16B): `Ri`
- Full Word (4B): `Ri.0...Ri.3`

The MAP program supports Big-Endian architecture:

- MSbit is #127, MS byte is #0, MS word is `Ri.0`
- LSbit is #0, LS byte is #15, LS word is `Ri.3`

4.4 Processing Memory (PMEM) - Structure Storage

The MAP supports the allocation of structures in its processing memory, the PMEM, which is a per-MAP pre-allocated RAM.

Structures are allocated using the STALLOC instruction. Internal registers maintain the status of allocated structures and their respective offsets in PMEM:

- `STR.PRESENT`: A 14b bitmap, a set bit indicates that the corresponding structure is allocated.
- `STR.OFFSET`: A 14B array, each byte indicates the offset of the corresponding structure in PMEM, in 4B granularity.

Up to 14 structures are supported. Struct-ID 0 is the **Standard Metadata (SMD)**, and is pre-allocated and initialized by the HW.

4.5 Global Data Storage

4.5.1 RAM Shared Area

The X-Switch HW supports the designation of a data storage area in RAM which is shared by all MAPs. All RAMs are initialized to 0 upon device reset. No memory initialization takes place afterwords (i.e. between packet processing sessions). The MAP can read/write from/to this area via special instructions (LDD/STD), using addresses relative to the base address of the global data storage.

4.5.2 Scratchpad Shared Area

The X-Switch HW also includes a scratchpad memory which is Closely-Coupled Memory (CCM), shared by all MAPs, which is initialized to 0 upon device reset. The MAP can read/write from/to this area via special instructions (LDSP/STSP).

4.6 MAP Initial Entry Points

The MAP program starts at one of the several entry points, associated with an ingress port type and different initial conditions. The entry points are specified by corresponding labels in the MAP program.

4.7 MAP Registers Pre-loading Scheme

The following MAP registers are preloaded by hardware when a MAP program is started:

- Registers `HDR.PRESENT (R11)` and `HDR.OFFSET0/1 (R12/R13)` are preloaded with the parsing data.
- Register `R7.0` is preloaded with SMD bytes 0 to 3.
- All general purpose registers (except for `R7.0`) are initialized to 0.

In addition, the parser program can flexibly initialize any of the MAP registers using **EXTMAP** and **MOVMAP** instructions. The MAP registers are initialized to 0 for each new packet. Then, the parser program can set any of them using the `EXTMAP` and `MOVMAP` instructions, after which, `R7.0` and `R11` to `R13`, are set as described above.

During MAP program execution, a packet can be *reparsed*. In this case, a different behavior is applied by the parser program. Refer to the REPARSE instruction, Section 4.13.46, and Section 3, Parser Programming Model for further details.

4.8 Arithmetic and Logical Operations

4.8.1 Signed and Unsigned Operations

By default, all values extracted from partial registers or from frame or structures, are treated as unsigned values. A special instruction prefix (**.SX**) can be used for sign-extending the result. When the sign-extend option is selected for an operation, the operands are treated as signed numbers.

4.8.2 Condition Code Flags: Z, N, C, V

The MAP supports condition code flags: Z (zero), N (negative), C (carry), and V (overflow). These flags are affected by several MAP instructions, and can be referred to by subsequent conditional branch operations.

In arithmetic and logic operations, operands are extracted (with or without sign-extend), and treated as 32b size operands. The following rules for setting the flags apply:

Table 4: Operation Condition Flags

Operation or Instruction	Example	Rules
Addition	D = A + B	**Z** = 1 when (D is 0).
		N = 1 when (D[31/15] is 1) (31/15 depends on operation size).
		C = 1 when result doesn't fit 32b/16b (D[32/16] is 1 if D were a 64b/32b operand).
		V = 1 ((D[31/15] is 1 and A[31/15] is 0 and B[31/15] is 0) *OR* (D[31/15] is 0 and A[31/15] is 1 and B[31/15] is 1)).
Subtraction, Compare	D = A - B	**Z** = 1 when (D is 0).
		N = 1 when (D[31/15] is 1) (31/15 depends on subtraction operation size).
		C = 1 on ***borrow*** (i.e., when A and B are treated as unsigned, and A < B)
		V = ((D[31/15] is 0 and A[31/15] is 1 and B[31/15] is 0) *OR* (D[31/15] is 1 and A[31/15] is 0 and B[31/15] is 1)).
AND/OR/XOR/ NOR/SHL/SHR[a,b]	D = A <op> B and **.F** is opted.	**Z** = 1 when (D is 0). **N** = 1 when (D[31] is 1). **C, V** flags are reset to 0.
FFI.F		**Z** =1 when a non-0 field was not found in the scanned value.

a. In SHL/SHR, the .F option is supported only in 4B mode.

b. In SHR instruction, the shift-by value may be equal to or greater than the shifted field size. In such a case, no data is written to the destination register, and Z flag is set (when .F is opted).

4.9 Lookup/Asynchronous Operation Flag (LFLAG)

Many asynchronous operations require a Lookup/Asynchronous Operation Flag (LFLAG) to query the operation status. HW supports eight LFLAGs (0 to 7). As such, eight instructions can be executed simultaneously.

An LFLAG is a two-bit status indicator, representing two aspects of the asynchronous operation:

- **Operation completion status:** Indicates whether the operation has completed or is still pending.
- **Operation result status:** Indicates whether the operation has completed successfully or failed.

Initially, when a MAP program starts, both bits of all LFLAGs are set by default to TRUE (that is, all operations completed, and completed successfully).

- When a lookup or another asynchronous operation is executed with an LFLAG, the HW initializes the completion-status to FALSE (at this point the result status is practically N/A).
- When the operation completes, completion-status is set to TRUE, and result status is set to TRUE/FALSE according to the result.
- When the MAP code executes **SYNC[.N] <LFLAG-bitmap>, PC-label**, the program stalls until completion status is TRUE, and then:
  - If result status is TRUE and **.N** is not specified, the program jumps to **PC-label**.
  - If result status is FALSE and **.N** is specified, the program jumps to **PC-label**.
  - Otherwise, the program continues to the next instruction.

> A SYNC instruction can be executed without an LFLAG (for example calling SYNC multiple times for the same LFLAG). In such cases, the LFLAG status remains as it was, and this LFLAG status determines how the instruction is executed.

4.10 Dependency Checker

The MAP supports a dependency checking mechanism that maintains a dependency list of registers, for which a write-back operation is pending. As a rule, every instruction that includes a destination register which is written with the result of the instruction, the destination register (or registers) is inserted into the dependency list, and is removed from the list when it is written to.

The purpose of placing registers in the dependency list is to protect against an attempt to read an instruction result before the result is ready; if the program attempts to read such a register, it will be stalled until the register is released from the dependency list, as a result of a write-back operation.

- In asynchronous operations (such as HASH, LKP, etc.), registers are inserted in the dependency list for a variable number of cycles. In many such operations, a **SYNC** instruction is used to wait for the completion of the operation.
  Therefore, an ***implicit SYNC*** can be performed by executing an instruction that reads the destination register.

  > If the program deliberately writes to the destination register, it is released from the dependency list even though a write-back in the background is pending.

- In synchronous operations (such as ADD/SUB, SHIFT, MOV, etc.), the instruction executes in a single cycle, however the write-back of the result is performed after one additional cycle. The destination register is therefore inserted in the dependency list for one cycle, creating a ***delay slot***. If the program executes such an instruction, and then attempts to read the result in the subsequent instruction, the dependency check will stall the program for one cycle. The delay slot can be exploited by an instruction that doesn't depend on the result of the previous operation.

The dependency checker supports the register addressing modes:

- When a 16B register is inserted in the dependency list, reading either the full register or any of its 4B parts causes the program to stall.
- When a 4B register is inserted in the dependency list, reading either the specific 4B part or the entire 16B register causes the program to stall.

A dependency checker for source registers is also supported. In several asynchronous operations, the source registers are not conveyed to the co-processor (COP) immediately, but rather sampled later when the COP is available to start the operation. The dependency checker protects from corruption of these source registers, by stalling the instruction which attempts to modify the register(s) until their value is conveyed to the COP.

4.11 Instruction Syntax Conventions

Table 5: Instruction Syntax Conventions

Acronym	Derivatives/Examples	Description
RS	RSS, RSE	Source register Derivatives: a range of {Start register, End register}
Rd	RdS, RdE	Destination register
Z		Zero flag
N		Negative flag
C		Carry flag
V		Overflow flag
<Field>	LKP.LF1, HASH.H2, R1.2	Field is an operand or instruction complement, e.g. LF<N>, where N = 0 to 7
[Opt]		Opt is an optional instruction extension: [.NP], [.F], [.H], [.SX], [.SH], [.CD],<.4Bsel>
[.NP]		Non-posted operation flag. Operations are *posted* by default. Relevant only for Counter, CAS/TAS, BW, and DLB instructions.
[.F]		Update ZNCV flags.
[.H]		Halt MAP program.
[.SX]		Perform sign-extend operation.
[.SH]		Perform a short (16b) arithmetic operation.
[.CD]		Clear destination register before the operation.
<.4Bsel>	R1.3, R7.0	.4Bsel = 4 Byte Select register word **.N**, where N=0 to 3 (optional); e.g., R1.3 is 3rd word of R1 register

4.12 Instruction Set

This section describes each MAP ISA instruction. A table summary is followed by detailed information for each instruction or instruction set.

Table 6: Instruction Summary (Sheet 1 of 2)

Category	Instruction	Operation	Details
Arithmetic and Logical	ADD	Add 2 numbers	Section 4.13.1
	SUB	Subtract 2 numbers	Section 4.13.2
	MOD	Modulo numbers	Section 4.13.3
	CMP	Compare numbers	Section 4.13.4
	AND	Bitwise AND	Section 4.13.5
	OR	Bitwise OR	Section 4.13.6
	XOR	Bitwise XOR	Section 4.13.7
	NOT	Bitwise NOT	Section 4.13.8
	SHL, SHR	Shift-left/right	Section 4.13.9
Bit/Byte Manipulation	CONCAT	Bitwise data concatenation	Section 4.13.10
	MOV	Move data	Section 4.13.11
	FFI	Find first non-zero bit	Section 4.13.12
Load/Store	LD	Load data	Section 4.13.13
	ST	Store data	Section 4.13.14
Jump/Branch	JTL	Jump by table index/bitmap	Section 4.13.15
	CALL	Call a subroutine	Section 4.13.16
	RET	Return from a subroutine	Section 4.13.17
	BR	Branch with/without condition	Section 4.13.18
Lookup	HASH	Calculate a Hash function on a Hash key	Section 4.13.19
	LKP	Perform table lookup	Section 4.13.20
Control	SYNC	Wait for an asynchronous operation result	Section 4.13.21
	HALT	Halt execution thread	Section 4.13.22
Atomic Operations	COUNTER	Increment/decrement/set a counter vector	Section 4.13.26
	METER	Activate a meter (policer)	Section 4.13.27
	CAS, TAS	Perform Compare-and-Swap or Test-and-Set atomic RAM operation	Section 4.13.28
	BW	Perform bitwise atomic RAM operation	Section 4.13.29
	DLB	Dynamic load balancing	Section 4.13.30

Table 6: Instruction Summary (Sheet 2 of 2)

Category	Instruction	Operation	Details
Miscellaneous	CP	Copy data to frame memory	Section 4.13.23
	CHKSUM	Calculate checksum	Section 4.13.24
	SEND	Send or drop packets	Section 4.13.25
	LDRTC	Load Real-Time Clock (RTC)	Section 4.13.31
	LDID	Load HW-ID	Section 4.13.32
	AQM	Adaptive Queue Management Query	Section 4.13.33
	RAND	Generate random number	Section 4.13.34
	STALLOC	Allocate a STRUCT	Section 4.13.35
	STRGET	Get STR.PRESENT and STR.OFFSET	Section 4.13.36
	STRSET	Set STR.PRESENT and STR.OFFSET	Section 4.13.37
	STRGETCUR	Get the STRUCT area cursor	Section 4.13.38
	STRSETCUR	Set the STRUCT area cursor	Section 4.13.39
	LBALLOC	Local buffer allocation and link	Section 4.13.40
	FFLUSH	Frame memory data flush	Section 4.13.41
	LBFREE	Free local buffer	Section 4.13.42
	FREBASE	Re-base frame memory buffer pointers	Section 4.13.43
	SETREFCNT	Set buffer reference count	Section 4.13.44
	GETFPTR	Get header buffer pointers and header size	Section 4.13.45
	REPARSE	Reparse packet header	Section 4.13.46
	IREQ	Interrupt request	Section 4.13.47
	IRET	Return from and interrupt or an exception	Section 4.13.48
	SWI	Soft interrupt	Section 4.13.49
	WAIT	Wait (stall the MAP)	Section 4.13.50
	NOP	No operation	Section 4.13.51
	SIZEQUERY	Perform packet size query	Section 4.13.52
	MCREQUEST	Request multicast resources	Section 4.13.53
	MCDONE	Multicast processing completed	Section 4.13.54

4.13 Instruction Details

The following subsections describe the details per instruction group.

The instruction syntax for all variations of the same instruction is given first. Then each variant of the instruction is described in its own sub-section of the general instruction.

4.13.1 ADD, ADDI

Function: Addition

Description

Add source operand 1 to source operand 2 and place the result in the destination register. Specific bit fields can be selected for the operation (by specifying field-offset and field-size operands). Immediate Value of 32b can be used as second source operand.

Any flag updates occur only if the set flags suffix (.F) is used.

Syntax

ADD:
```
ADD[.F][.SX][.SH] Rd.<4Bsel>, Rs1.<4Bsel>, Source1StartOffset,
Source1Size, Rs2.<4Bsel>, Source2StartOffset, Source2Size
```

ADDI:
```
ADDI[.F][.SX][.SH] Rd.<4Bsel>, Rs1.<4Bsel>, Source1StartOffset,
Source1Size, ImmediateValue
```

Instruction Format

ADD: ADD[.F][.SX][.SH] Dest, Source1, Source1StartOffset, Source1Size, Source2, Source2StartOffset, Source2Size

Instruction Operation: Dest[N:0] = Source1[i1:j1] + Source2[i2:j2]

- **.SH:** A short (16b) operation. By default, the normal operation is 32b.
- **Dest[N:0]:** Normal operation: N=31;
  Short operation: N=15.
- Source operands can be a maximum of 4 bytes.
- **j:** StartOffset. Range: 0 to 31. Units: Bits.
- **i:** StartOffset + Size. Units: Bits.
  Size max:
  - Short operation: 16.
  - Normal operation: 32.
- **.SX:** Sign-extends the result in the destination register.
  - When this option is selected, the operands are treated as signed integers.
  - When this option is not selected, the operands are treated as unsigned integers.

Instruction Format

ADDI: ADDI[.F][.SX][.SH] Dest, Source1, Source1StartOffset, Source1Size, ImmediateValue

Instruction Operation: Dest[N:0] = Source1[i1:j1] + ImmediateValue

- **.SH:** A short (16b) operation. By default, the normal operation is 32b.
- **Dest[N:0]:** - Normal operation: N=31.
  - Short operation: N=16.
- Source1 operand can be a maximum of 4 bytes.
- ImmediateValue: 16-bit value maximum.
- **j:** StartOffset. Units: Bits.
- **i:** StartOffset + Size. Units: Bits.

Timing Characteristics: Single Cycle

Flags Affected: (when [.F] is opted)

Refer to Section 4.8.2, Condition Code Flags: Z, N, C, V.

Pseudo Code

```
Dest = Rd[.<4B-sel>]
Src1 = Rs1[.<4B-sel>][field1_offset+field1_size:field1_offset]
Src2 = Rs2[.<4B-sel>][field1_offset+field1_size:field1_offset]
(For ADDI: Src2 = Immediate_value)
If .F:
  Flag Z = (Dest == 0)
  Flag N = (Dest[MSbit] == 1)
  Flag C = ...
  Flag V = ...
```

4.13.2 SUB, SUBI

Function: Subtraction

Syntax

SUB:	`SUB[.F][.SX][.SH]  Rd.<4Bsel>, Rs1.<4Bsel>, Source1StartOffset, Source1Size, Rs2.<4Bsel>, Source2StartOffset, Source2Size`
SUBI:	`SUBI[.F][.SX][.SH]  Rd.<4Bsel>, Rs1.<4Bsel>, Source1StartOffset, Source1Size, ImmediateValue`

Instruction Format

SUB: SUB Dest, Source1, Source1StartOffset, Source1Size, Source2, Source2StartOffset, Source2Size

Instruction Operation: Dest[N:0] = Source1[i1:j1] - Source2[i2:j2]

• **.SH:**	A short (16b) operation. By default, the normal operation is 32b.
• **Dest[N:0]:**	Normal operation: N=31.
	Short operation: N=15.
• Sources max:	4 bytes
• **j:** StartOffset.	Units: Bits.
• **i:** StartOffset + Size.	Units: Bits.

Instruction Format

SUBI: SUBI Dest, Source1, Source1StartOffset, Source1Size, ImmediateValue

Instruction Operation: Dest[N:0] = Source1[i1:j1] - ImmediateValue

• Source1 max:	4 bytes.
• ImmediateValue:	Max: 16-bit value
• **j:** StartOffset.	Units: Bits.
• **i:** StartOffset + Size.	Units: Bits.
• **.SH:**	A short (16b) operation. By default, the normal operation is 32b.
• **Dest[N:0]:**	Normal operation: N=31.
	Short operation: N=15. (16-MSbits are left intact.)

- **.SX:** Sign-extends the result in the destination register.
  - When this option is selected, the operands are treated as signed integers.
  - When this option is not selected, the operands are treated as unsigned integers.

Timing Characteristics: Single Cycle

Flags Affected: **(when [.F] is opted)**

Refer to Section 4.8.2, Condition Code Flags: Z, N, C, V.

4.13.3 MOD, MODI

Function: Modulu operation

Syntax

MOD:
```
MOD[.LB]   Rd.<4Bsel>, Rs1.<4Bsel>, Source1StartOffset,
Source1Size, Rs2.<4Bsel>, Source2StartOffset, Source2Size
```

MODI:
```
MODI[.LB]  Rd.<4Bsel>, Rs1.<4Bsel>, Source1StartOffset,
Source1Size, ImmediateValue
```

Instruction Format

MOD: MOD [.LB]Dest, Source1, Source1StartOffset, Source1Size, Source2, Source2StartOffset, Source2Size

Instruction Operation: Dest = Source1[i1:j1] MOD Source2[i2:j2]

Instruction Format

MODI: MODI[.LB] Dest, Source1, Source1StartOffset, Source1Size, ImmediateValue

Instruction Operation: Dest = Source1[i1:j1] MOD ImmediateValue

- **.LB:** The HW supports two types of Modulus operations:
  - *Arithmetic*: the default operation.
  - *Load-balancing*: A latency-optimized implementation of multiplier-based MOD operation is performed. This MOD type is performed when **.LB** is opted.
    In this option, the maximum size of **Source2** is 13b:
    - **Source2Size**: Size in bits of Source2. Range: 1 to 13.
    - **ImmediateValue:** 1 to 8K (up to 13b value).
- **Source1Size:** Size in bits of Source1. Range: 1 to (32 - Source1StartOffset).
- **Source2Size:** Size in bits of Source2. Range: 1 to 18.
- **ImmediateValue:** 1 to 256K (up to 18b value)
- **j:** StartOffset
- **i1:** Source1StartOffset + Source1Size - 1
- **i2:** Source2StartOffset + Source2Size - 1

Timing Characteristics: Asynchronous operation

4.13.4 CMP, CMPI

Function: Compare

Syntax

CMP:
```
CMP  Rs1.<4Bsel>, Source1StartOffset, Rs2.<4Bsel>,
Source2StartOffset, Size
```

CMPI:
```
CMPI  Rs1.<4Bsel>, Source1StartOffset, ImmediateValue, Size
```

Instruction Format

CMP: CMP Source1, Source1StartOffset, Source2, Source2StartOffset, SizeBits

Instruction Operation: Result = Source1[i1:j1] - Source2[i2:j2]

- **Source1, Source2:** 4B registers
- **SizeBits:** 1 to 32
- **j:** StartOffset. Units: Bits
- **i:** StartOffset + Size. Units: Bits - 1
- **Source1** and **Source2** are considered unsigned numbers.
- **Z** flag = (Result == 0): Source1 and Source2 are equal.
- **C** flag = (Result < 0): Source1 is less than Source2.

See also Section 4.8.2, Condition Code Flags: Z, N, C, V

Instruction Format

CMPI: CMPI Source1, Souce1StartOffset, ImmediateValue, SizeBits

Instruction Operation: Result = Source1[i1:j1] - ImmediateValue

- **ImmediateValue:** 16b value
- **SizeBits:** 1 to 32

> When *Size* is greater than 16b, **ImmediateValue**[*Size*-1:16] is padded with 0's.

- **Source1:** 4B register
- **Source1** and **ImmediateValue** are considered unsigned numbers.
- **j:** StartOffset. Units: Bits (0 to 31)
- **i:** StartOffset + Size. Units: Bits
- **Z** flag = (Result == 0): Source1 and ImmediateValue are equal.
- **C** flag = (Result < 0): Source1 is less than ImmediateValue.

See also Section 4.8.2, Condition Code Flags: Z, N, C, V.

Timing Characteristics: Single Cycle

Flags Affected: See **Instruction Operation** description above.

4.13.5 AND, ANDI

Function: Bitwise AND

Syntax

AND: `AND[.F]  Rd.<4Bsel>, Rs1.<4Bsel>, Src1StartOffset, Rs2.<4Bsel>, Src2StartOffset, Size`

ANDI: `ANDI[.F] Rd.<4Bsel>, Rs1.<4Bsel>, Src1StartOffset, ImmediateValue, Size`

Instruction Format

AND: AND, Dest, Source1, Source1StartOffset, Source2, Source2StartOffset, Size

Instruction Operation: Dest[Size-1:0] = Source1[i1:j1] & Source2[i2:j2]

- Dest, Source1 and Source2 are 4B registers.
- StartOffset values: 0 to 31; Size value: 1 to 32.
- **j:** StartOffset. Units: Bits
- **i:** StartOffset + Size. Units: Bits

Only *Size* bits are copied to the Dest register, the remainder of the register is left intact.

Instruction Format

ANDI: ANDI Dest, Source1, Src1StartOffset, ImmediateValue, Size

Instruction Operation: Dest[Size-1:0] = Source1[i1:j1] & ImmediateValue

Only 2B mode is supported

- **ImmediateValue:** 16b value
- **SizeBits:** 1 to 32

When *Size* is greater than 16b, **ImmediateValue**[*Size*-1:16] is padded with 0's.

- **Source1:** 4B register
- **j:** StartOffset. Units: Bits (0 to 31)
- **i:** StartOffset + Size Units: Bits

Only *Size* bits are copied to the Dest register, the remainder of the register is left intact.

Timing Characteristics: Single Cycle

Flags Affected: **(when [.F] is opted)**

AND.F can be used without destination (Rd = R15/RN (null)), in which case the operation affects only the Z flag. Refer to Section 4.8.2, Condition Code Flags: Z, N, C, V.

4.13.6 OR, ORI

Function: Bitwise OR

Syntax

OR: `OR[.F]  Rd.<4Bsel>, Rs1.<4Bsel>, Src1StartOffset, Rs2.<4Bsel>, Src2StartOffset, Size`

ORI: `ORI[.F]  Rd.<4Bsel>, Rs1.<4Bsel>, Src1StartOffset, ImmediateValue, Size`

Instruction Format

OR: OR Dest, Source1, Source1StartOffset, Source2, Source2StartOffset, Size

Instruction Operation: Dest[Size-1:0] = Source1[i1:j1] | Source2[i2:j2]

- Dest, Source1 and Source2 are 4B registers.
- StartOffset values are 0 to 31, Size value is 1 to 32.
- **j:** StartOffset. Units: Bits.
- **i:** StartOffset + Size. Units: Bits

Only *Size* bits are copied to the Dest register, the remainder of the register is left intact.

Instruction Format

ORI: ORI Dest, Source1, Src1StartOffset, ImmediateValue, Size

Instruction Operation: Dest[Size-1:0] = Source1[i1:j1] | ImmediateValue

Only 2B mode is supported.

- **ImmediateValue:** 16b value
- **SizeBits:** 1 to 32

When *Size* is greater than 16b, **ImmediateValue**[*Size*-1:16] is padded with 0’s.

- **Source1:** 4B register
- **j:** StartOffset. Units: Bits (0 to 31)
- **i:** StartOffset + Size. Units: Bits

Only *Size* bits are copied to the Dest register, the remainder of the register is left intact.

Timing Characteristics: Single Cycle

Flags Affected: (when [.F] is opted)

Refer to Section 4.8.2, Condition Code Flags: Z, N, C, V. **OR.F** can be used without destination (Rd = R15/RN (null)), in which case the operation affects only the Z flag.

4.13.7 XOR, XORI

Function: Bitwise XOR

Syntax

XOR: `XOR[.F]  Rd.<4Bsel>, Rs1.<4Bsel>, Src1StartOffset, Rs2.<4Bsel>, Src2StartOffset, Size`

XORI: `XORI[.F]  Rd.<4Bsel>, Rs1.<4Bsel>, Src1StartOffset, ImmediateValue, Size`

Instruction Format

XOR: XOR Dest, Source1, Source1StartOffset, Source2, Source2StartOffset, Size

Instruction Operation: Dest[Size-1:0] = Source1[i1:j1] ^ Source2[i2:j2]

- Dest, Source1 and Source2 are 4B registers.
- StartOffset values: 0 to 31;
  Size value: 1 to 32.
- **j:** StartOffset. Units: Bits.
- **i:** StartOffset + Size. Units: Bits

> Only *Size* bits are copied to the Dest register, the remainder of the register is left intact.

Instruction Format

XORI: XORI Dest, Source1, Src1StartOffset, ImmediateValue, Size

Instruction Operation: Dest[Size-1:0] = Source1[i1:j1] ^ ImmediateValue

Only 2B mode is supported.

- **ImmediateValue:** 16b value
- **SizeBits:** 1 to 32

> When *Size* is greater than 16b, **ImmediateValue**[*Size*-1:16] is padded with 0's.

- **Source1:** 4B register
- **j:** StartOffset Units: Bits (0 to 31).
- **i:** StartOffset + Size. Units: Bits.

> Only *Size* bits are copied to the Dest register, the remainder of the register is left intact.

Timing Characteristics: Single Cycle

Flags Affected: **(when [.F] is opted)**

Refer to Section 4.8.2, Condition Code Flags: Z, N, C, V. **XOR.F** can be used without destination (Rd = R15/RN (null)), in which case the operation affects only the Z flag.

4.13.8 NOT

Function: Bitwise Negate

Syntax

NOT: `NOT[.F] Rd.<4Bsel>, Rs.<4Bsel>, SourceStartOffset, SourceSize`

Instruction Format

NOT: Not Dest, Source, SourceStartOffset, SourceSize

Instruction Operation: Dest[Size-1:0] = Negate(Source[i:j])

- Dest, Source1 and Source2 are 4B registers.
- StartOffset values: 0 to 31;
  Size value: 1 to 32.
- **j:** StartOffset. Units: Bits
- **i:** StartOffset + Size. Units: Bits

> Only *SourceSize* bits are copied to the Dest register,
> the remainder of the register is left intact.

Timing Characteristics: Single Cycle

Flags Affected: (when [.F] is opted)

Refer to Section 4.8.2, Condition Code Flags: Z, N, C, V. **NOT.F** can be used without destination (Rd = R15/RN (null)), in which case the operation affects only the Z flag.

4.13.9 SHL, SHLI, SHR, SHRI

Function: Bitwise shift left/right.

Syntax

SHL:	`SHL[.F][.CD] Rd[.<4Bsel>], Rs1[.<4Bsel>], Source1StartOffset, Source1Size, Rs2.<4Bsel>, Source2StartOffset, Source2Size`
SHLI:	`SHLI[.F][.CD] Rd[.<4Bsel>], Rs[.<4Bsel>], Source1StartOffset, Source1Size, ShiftImmediateValue`
SHR:	`SHR[.F][.CD] Rd[.<4Bsel>], Rs1[.<4Bsel>], Source1StartOffset, Source1Size, Rs2.<4Bsel>, Source2StartOffset, Source2Size`
SHRI:	`SHRI[.F][.CD] Rd[.<4Bsel>], Rs[.<4Bsel>], Source1StartOffset, Source1Size, ShiftImmediateValue`

Instruction Format

SHL: SHL[.F][.CD] Dest, Source1, Source1StartOffset, Source1Size, Source2, Source2StartOffset, Source2Size

SHLI: SHLI[.F][.CD] Dest, Source1, Source1StartOffset, Source1Size, ImmediateValue

SHR: SHR[.F][.CD] Dest, Source1, Source1StartOffset, Source1Size, Source2, Source2StartOffset, Source2Size

SHRI: SHRI[.F][.CD] Dest, Source1, Source1StartOffset, Source1Size, ImmediateValue

Instruction Operation:

SHL: Dest[m:k] = Source1[i1:j1] << Source2[i2:j2]

SHR: Dest[m:0] = Source1[i1:j1] >> Source2[i2:j2]

- **j:** StartOffset. Units: Bits
- **i:** StartOffset + Size. Units: Bits
- **m, k:** determined by the shift value,
  see SHR Illustrated Operation and SHL Illustrated Operation.
  - **k** = value in SrcReg2[i2:j2]
  - **m**
    - For **SHL**: **m**= **k** + Source1Size
    - For **SHR**: **m**= Source1Size - **k**

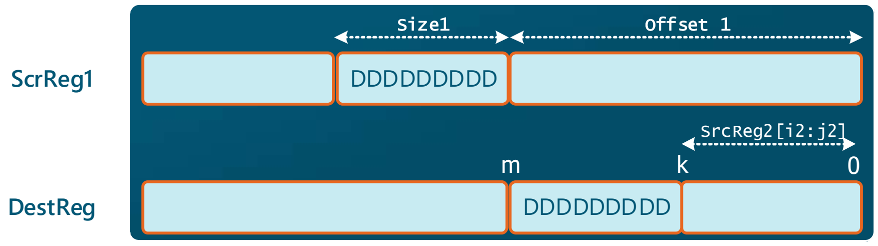
> If m≤0, no value is written to the destination register, however, the register is released by the dependency checker, and Z flag is set (when .F is opted).

There are 2 modes of operation, 16B and 4B:

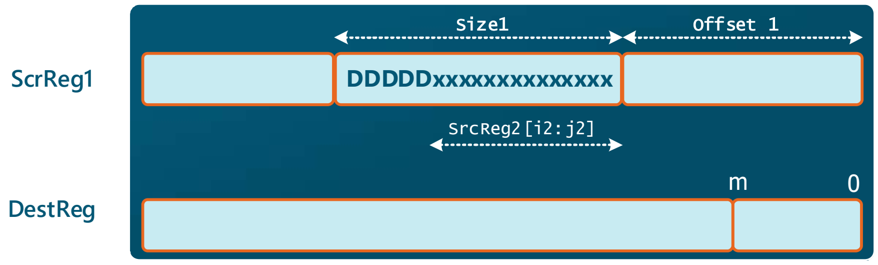
- **16B:** Asynchronous operation in COP, supports up to 128b:
    - **Dest, Source1:** Full 16B register.
    - **Source2:** 4B register.
    - **Source1StartOffset:** 0 to 127.
    - **Source1Size:** 1 to 128.
    - **Source2StartOffset:** 0 to 31.
    - **Source2Size:** 1 to 7.
    - **ShiftImmediateValue:** 0 to 127.
- **4B:** Synchronous operation, supports up to 32b:
    - **Dest, Source1, Source2:** 4B registers.
    - **Source1Offset, Source2Offset:** 0 to 31.
    - **Source1Size:** 1 to 32.
    - **Source2Size:** 1 to 5.
    - **ShiftImmediateValue:** 0 to 31.
    - **.F:** Option is relevant only for 4B mode.
    - **.CD:** Clear-destination before assignment.

> If not selected: only written bits in DestReg are modified.

SHL Illustrated Operation



SHR Illustrated Operation



Timing Characteristics:

- **16B mode:** Asynchronous operation. Pseudo-SYNC is performed by reading the Dest register
- **4B mode:** Single cycle

Flags Affected: (when [.F] is opted):
Refer to Section 4.8.2, Condition Code Flags: Z, N, C, V.

4.13.10 CONCAT

Function: Bitwise concatenation.

Syntax

CONCAT: `CONCAT[.CD] Rd.<4Bsel>, DestOffset, Rs1.<4Bsel>, Src1StartOffset, Src1Size[, Rs2.<4Bsel>, Src2StartOffset, Src2Size]`

Instruction Format

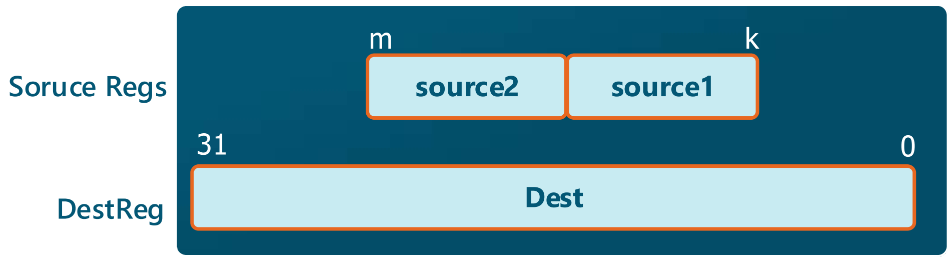
CONCAT: CONCAT[.CD] DestReg, DestOffset, Src1Reg, Src1StartOffset, Src1Size[, Src2Reg, Src2StartOffset, Src2Size]

Instruction Operation:

- For 2 sources: DestReg[m:k] = (Src2Reg[i2:j2] << Src1Size) | Src1Reg[i1:j1]
- For 1 source: DestReg[m:k] = Src1Reg[i1:j1]
- **DestReg, Src1Reg and Src2Reg** Register Sizes: 4B.
- **Src1Size, Src2Size** **Range:** 1 to 32. Units: Bits.
  Src1Size + Src2Size + DestOffset must be less than or equal to 32.
- **DestOffset, Src1StartOffset, Src2StartOffset: Range:** 0 to 31. **Units:** Bits.
- **j:** StartOffset
- **i:** (StartOffset + Size - 1)
- **k:** DestOffset
- **m:** (DestOffset + Src1Size + Src2Size - 1)
  Where ***m*** must be < 32.
- **.CD:** Clear DestReg before placing the result.
  When .CD is not opted, only the written bits are affected; all other bits remain unchanged.

> See figure below for `Src1` and `Src2` placement.

Illustrated Operation:



Timing Characteristics: Single Cycle

4.13.11 MOV, MOVI

Function: Move data to a register.

Syntax

MOV: `MOV[.CD]  Rd.<4Bsel>, Rs.<4Bsel>`

MOVI: `MOVI[.CD]  Rd.<4Bsel>, SourceImmediateValue`

Instruction Format

MOV: MOV[.CD] DestReg, SourceReg

- DestReg, SourceReg, are 4B registers.
- If **.CD** is opted, the entire 16B DestReg is cleared.

> To extract a numeric field with sign-extend, `SHLI` can be used with shift-size 0.

Instruction Format

MOVI: MOVI[.CD] DestReg, SourceImmediateValue

Instruction Operation: Copy from source register (or immediate value) to destination register

- **SourceImmediateValue:** Max: 32b value
- If **.CD** is opted, the entire 16B DestReg is cleared.

Timing Characteristics: Single Cycle

4.13.12 FFI

Function: Find first non-zero field.

Syntax

FFI: `FFI.F Rd.<4Bsel>, Rs1.<4Bsel>, Rs2.<4Bsel>, FieldSizeImm, ScanDirectionImm`

Instruction Format

FFI.F DestReg, ValueReg, OffsetReg, FieldSizeImm, ScanDirectionImm

Instruction Operation: Find the first non-zero field in a 32b value and return the offset and content of the field. Scan direction can be either MSbit to LSbit or vice versa.

The field size can be 1-4 bits.

• **If found:**	Field offset and content are retrieved and placed in **DestReg**, and Z flag is cleared.
• **If *not* found:**	Z flag is set.
• **ValueReg**:	A 32b value to scan.
• **OffsetReg[4:0]:**	The Start scan offset. Units: Bits. The bit offset to start searching for a non-0 field. Range: 0 - 31.

> Offset must be aligned to field size.
>
> - When direction is LSbit to MSbit, scan start offset = OffsetReg[4:0] = N× field_size
> - When direction is MSbit to LSbit, scan start offset = OffsetReg[4:0] = N× field_size - 1

- **DestReg:**
  - **DestReg[4:0]:** field offset (0-31 if found). Units: Bits.
  - **DestReg[11:8]:** field content (if found).
- **FieldSizeImm**: 1/2/3/4 Bits.
- **ScanDirectionImm**: **0**: MSb to LSb; **1**: LSb to MSb

  Example: Field size = 3
  - For LSb toMSb: Assume scan start is 0:
    - **DestReg[4:0]: Field** Offset is N.
    - **DestReg[11:8] = ValueReg[MIN(31, N+2):N].**
  - For MSb to LSb: assume scan start is 29:
    - Scan starts from **ValueReg[29:27]** and downwards.
    - **DestReg[4:0]** = field offset is N.
    - **DestReg[11:8]** =**ValueReg[(N+2):N]**

Timing Characteristics: Single Cycle

Flags Affected: (when [.F] is opted.)

Refer to Section 4.8.2, Condition Code Flags: Z, N, C, V.

4.13.13 LD, LDD, LDDI, LDH, LDS, LDSP, LDSPI

Function: Load data from RAM or scratchpad memory.

Syntax

LD:	`LD    RdS[.<4Bsel>], RdE[.<4Bsel>], Raddr.<4Bsel>, SizeBytes`
LDD:	`LDD   RdS[.<4Bsel>], RdE[.<4Bsel>], Raddr.<4Bsel>, SizeBytes`
LDDI:	`LDDI  RdS[.<4Bsel>], RdE[.<4Bsel>], DirectAddrImm, SizeBytes`
LDH:	`LDH   Rd[.<4Bsel>], HeaderOffsetID, AdditionalOffsetImm, SizeBytes`
LDS:	`LDS   Rd[.<4Bsel>], StructID, AdditionalOffsetImm, SizeBytes`
LDSP:	`LDSP  Rd.<4Bsel>, Raddr.<4Bsel>`
LDSPI:	`LDSPI Rd.<4Bsel>, DirectAddrImm`

Instruction Format

LD: LD DestStartReg, DestEndReg, SourceReg, SizeBytes

Instruction Operation: Load data from RAM to destination.

- **SourceReg[31:0]:** Specify the absolute RAM address.
- **SizeBytes:** Specifies the amount of data to read. Range: 4B, 8B, 16B, or 32B.

> The data read from RAM must reside within a 32B line,
> that is: (Address & 0xFFFFE0) = ((Address + size - 1) & 0xFFFFE0)

- **DestStart, DestEnd:** Destination register(s).
  - For 4B register: DestStart and DestEnd must be identical.
  - For 4B≤SizeBytes≤16B: Specify a single 16B register (identical for both DestStart and DestEnd).

Instruction Format

LDD: LDD DestStartReg, DestEndReg, SourceReg, SizeBytes

LDDI: LDDI DestStartReg, DestEndReg|, SourceImm, SizeBytes

Instruction Operation: Load data from the global data section (GDMR) in RAM to destination.

- Offset address in the global data section is specified in SourceReg/SourceImm.
- The RAM address is calculated by **GDMR** + SourceReg
- **SizeBytes:** Specifies the amount of data to read. Range: 4B, 8B, 16B or 32B

> The data read from RAM must reside within a 32B line,
> that is: (Address & 0xFFFFE0) = ((Address + size - 1) & 0xFFFFE0)

- **DestStart, DestEnd:** Destination register(s).
  - For 4B register: DestStart and DestEnd must be identical.
  - For 4B≤SizeBytes≤16B: Specify a single 16B register (identical for both DestStart and DestEnd).

Instruction Format

LDH: LDH DestReg, HeaderOffsetID, AdditionalOffsetImm, SizeBytes

Instruction Operation:

Load data from header location indicated by HeaderOffsetID (i.e., depending on `HeaderOffsetID` size:

- ❍ `HDR.OFFSET0[HeaderOffsetID]` *for* `HeaderOffsetID`<16
- ❍ `HDR.OFFSET1[HeaderOffsetID - 16]` *for* `HeaderOffsetID`≥16 + `AdditionalOffsetImm`.

• **SizeBytes:**	Amount of data to read. Range: 1B to 16B.
• **AdditionalOffsetImm:**	Additional offset from Header offset respective to `HeaderOffsetID`. Range: 0 - 255.

> Total HeaderOffset + AdditionalOffsetImm + SizeBytes must not exceed the header size. It is the MAP program's responsibility to ensure that the actual packet data is available in the accessed header offset. Otherwise a Header violation error is incurred.

Instruction Format

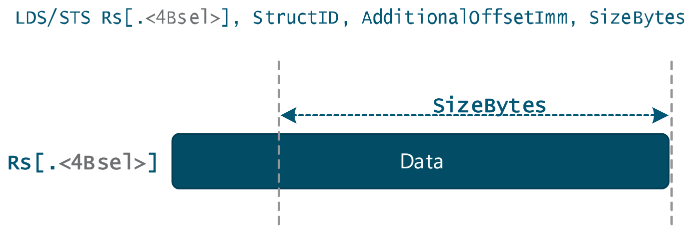
LDS: LDS DestReg, StructID, AdditionalOffsetImm, SizeBytes

Instruction Operation:

Load data from PMEM location indicated by STR.OFFSET[StructID] + AdditionalOffsetImm.

- **SizeBytes:** Amount of data to read. Range: 1B to 16B.

  The data read from the PMEM is placed in bits DestRegister[8 × (SizeBytes)-1:0]:


LDS/STS Rs[.<4Bsel>], StructID, AdditionalOffsetImm, SizeBytes
SizeBytes
Rs[.<4Bsel>]
Data

Instruction Format

LDSP: LDSP DestReg, AddrReg

LDSPI: LDSP DestReg, AddrImm

Instruction Operation:

Load 4B data from scratchpad memory to destination.

- **AddrReg[11:0]/AddrImm:** Byte address, must be 4B aligned (i.e., bits [1:0] of the address are ignored).

Timing Characteristics:

- **LD/LDD/LDH/LDS:** Asynchronous operation.
- **LDSP:** Synchronous operation

4.13.14 ST, STD, STDI, STH, STS, STSP, STSPI

Function: Store data in RAM or scratchpad memory.

Syntax

ST:	`ST[.SYNC]   Rs[.<4Bsel>], Raddr.<4Bsel>, SizeBytes`
STD:	`STD[.SYNC]  Rs[.<4Bsel>], Raddr.<4Bsel>, SizeBytes`
STDI:	`STDI[.SYNC] Rs[.<4Bsel>], DirectAddrImm, SizeBytes`
STH:	`STH[.SYNC]  Rs[.<4Bsel>], HeaderOffsetID, AdditionalOffsetImm, SizeBytes`
STS:	`STS[.SYNC]  Rs[.<4Bsel>], StructID, AdditionalOffsetImm, SizeBytes`
STSP:	`STSP       Rs.<4Bsel>, Raddr.<4Bsel>`
STSPI:	`STSPI      Rs.<4Bsel>, DirectAddrImm`

Instruction Format

ST: ST[.SYNC] SourceReg, AddrReg, SizeBytes

Instruction Operation: Store data from source to device memory.

- **AddrReg[31:0]:** The absolute device memory address is specified.
- **SizeBytes:** Specifies the amount of data to write. Range: 4B, 8B, or 16B.

> The data stored in device memory must reside within a 32B line,
> that is: (Address & 0xFFFFE0) = ((Address + size - 1) & 0xFFFFE0)

- **.SYNC:** When this option is selected, the instruction synchronizes the store operation, as well as any previous store operation, at the next asynchronous operation.

Instruction Format

STD: STD[.SYNC] SourceReg, AddrReg, SizeBytes

STDI: STDI[.SYNC] SourceReg, AddrImm, SizeBytes

Instruction Operation: Store data from source to global data section in device memory.

- The offset address in the global data section is specified in AddrReg/AddrImm value.
- **SizeBytes:** Specifies the amount of data to write. Range: 4B, 8B, or 16B.

> The data stored in device memory must reside within a 32B line,
> that is: (Address & 0xFFFFE0) = ((Address + size - 1) & 0xFFFFE0)

- **.SYNC:** When this option is selected, the instruction synchronizes the store operation, as well as any previous store operation, at the next asynchronous operation.

Instruction Format

STH: STH[.SYNC] SourceReg, HeaderOffsetID, AdditionalOffsetImm, SizeBytes

Instruction Operation:
Store data from source to header location indicated by `HeaderOffsetID`
(i.e., depending on `HeaderOffsetID` size:

- `HDR.OFFSET0[HeaderOffsetID]`   for `HeaderOffsetID`<16
- `HDR.OFFSET1[HeaderOffsetID-16]`   for `HeaderOffsetID`≥16) + `AdditionalOffsetImm`.

- **.SYNC:** When this is option is selected, the instruction synchronizes on the store operation, as well as any previous store operation.
- **SizeBytes:** Amount of data to write. Range: 1B to 16B.
- **AdditionalOffsetImm:** Additional offset from Header offset respective to `HeaderOffsetID`. Range: 0 - 255.

> Total HeaderOffset + AdditionalOffsetImm + SizeBytes must not exceed the header size. It is the MAP program’s responsibility to ensure that written data is within the packet data size.Otherwise a Header violation error is incurred.

Instruction Format

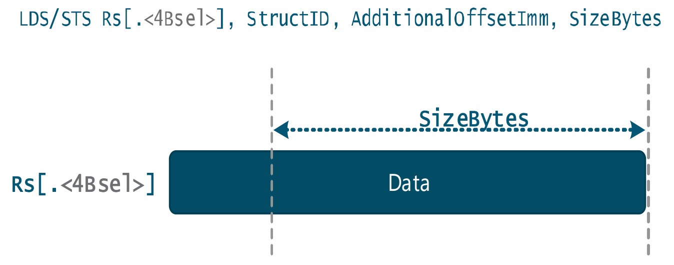
STS: STS[.SYNC] SourceReg, StructID, AdditionalOffsetImm, SizeBytes

Instruction Operation: Store data from source to PMEM location indicated by STR.OFFSET[StructID] + AdditionalOffsetImm (Units: Bytes).

> Data is written unconditionally, i.e. StructID is expected to be valid.

- **.SYNC:** When this option is selected, the instruction synchronizes the store operation, as well as any previous store operation, at the next asynchronous operation.
- **SizeBytes:** Amount of data to write. Range: 1B to 16B.

The data stored to PMEM is taken from bits DestRegister[8 × (SizeBytes)-1:0]:


LDS/STS Rs[.<4Bsel>], StructID, AdditionalOffsetImm, SizeBytes
SizeBytes
Rs[.<4Bsel>]
Data

Instruction Format

STSP: STSP SrcReg, AddrReg

STSPI: STSPI SrcReg, AddrImm

Instruction Operation: Store 4B data to scratchpad memory.

- **AddrReg[11:0]/AddrImm:** Byte address.
  Must be 4B aligned (i.e. address bits [1:0] are ignored).

Timing Characteristics:

ST/STD/STH/STS: Asynchronous operation. When **[.SYNC]** is opted, the instruction synchronizes on the store operation, as well as any previous store operation.

STSP: Synchronous operation

4.13.15 JTL

Function: Jump-table: Iterate over bitmap and call subroutines.

Syntax

`JTL[.NM] Rs.<4Bsel>, Rret.<4Bsel>, PC0 [, PC1 [, PC2 [, PC3 [, PC4]]]]] [, PCNoMatch]`

- **.NM:** Last PC in jump-table is a valid no-match address.

Instruction Format

JTL[.NM] Source, RetAddrReg, PC0, [..., PC4] [, PCNoMatch]

Instruction Operation: A jump table includes up to 5 entries [PC0,..,PC4], and a minimum of 2 entries. The RetAddrReg (`Rret.<4Bsel>`) is used by hardware to store the return address; the source includes a bitmap by which the jump address is selected:

- **Source[4:0]**: `Rs.<4Bsel>`
  Contains the bitmap to be used for the jump table
  (HW refers to these bits only).
- For each bit ***i*** in the bitmap, if bit is set:
  - Clear bit ***i:*** The bit is cleared in the source register.
  - Push the current PC to RetAddrReg
  - Jump to subroutine specified in PC<**i**>:
    The subroutine returns `RET RetAddrReg.`
- If the jump-to address is a subroutine, it returns with `RET RetAddrReg`.

  If there is no need to return, `RetAddrReg` can be R15/RN (null register), and the bitmap in Source register is not updated (no bit is cleared).
- When **.NM**
  - ***is opted***, a no-match jump-address is specified for cases where the relevant bits in the bitmap are zero.
  - ***is not opted*** and the relevant bits in the bitmap are clear, no jump is performed, and execution continues from the next instruction following the jump-table.

Timing Characteristics: Multiple Cycles

4.13.16 CALL

Function: Call subroutine

Syntax

```
CALL Rret<.4Bsel> PC
```

Instruction Format

CALL: CALL RetAddrReg, PCSubRoutine

Instruction Operation: Push next line's PC to RetAddrReg and jump to PCSubRoutine

- PCSubRoutine: A subroutine that ends with `RET RetAddrReg.`

Timing Characteristics: Multiple Cycles

4.13.17 RET

Function: Return from subroutine.

Syntax

```
RET Rret<.4Bsel>
```

- Rret includes the return address.

Instruction Format

RET: RET RetAddrReg

Instruction Operation:

Return from a subroutine, the return address is included in RetAddrReg.

Timing Characteristics: Single Cycle

4.13.18 BR, BRI, BRBTST

Branch with/without condition.

Syntax

BRcc: `BR[<condition-code>]  Rs.<4Bsel>`

BRIcc: `BRI[<condition-code>] PC`

BRBTSTcc:`BRBTST<condition-code> Rs.<4Bsel>, BitOffset, PC`

- <condition codes>

Instruction Format: See Operation below.

Instruction Operation: Branch with/without condition. The branch address provided in a register is absolute, while branch address in immediate value is relative.

- <**condition codes**>
  - **BREQ:** Branch if Z = 1 (*branch if equal*)
  - **BRNEQ:** Branch if Z = 0 (*branch if not equal*)
  - **BRLT:** Branch if N = 1 (*branch if less than*)
  - **BRGT:** Branch if N = 0 and Z = 0 (*branch if higher than*)
  - **BRGE:** Branch if N = 0 (*branch if greater or equal*)
  - **BRLE:** Branch if N = 1 or Z = 1 (*branch if less or equal*)
  - **BRC:** Branch when C = 1
  - **BRNC:** Branch when C = 0
  - **BRV:** Branch when V = 1
  - **BRNV:** Branch when V = 0
  - **BR:** Branch unconditionally
- Branch by bit-test:
  - **BitOffset:** 0 - 31
  - **BRBTSTSET:** Branch if Rs[BitOffset] is set.
  - **BRBTSTCLR:** Branch if Rs[BitOffset] is clear.
- **BRIcc:** Branch with immediate address.
  <**cc**> can be any of the above supported conditions.

Timing Characteristics: Single Cycle

4.13.19 HASH

Function: Generate a hash function.

Syntax

HASH: `HASH.H<N>[.W] Rd.<4Bsel>, RsS, RsE, [ Rs2.<4Bsel>,] SrcSizeBytes, HashSizeBits`

Instruction Format

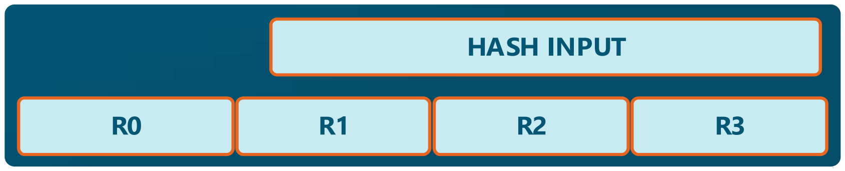
HASH: HASH.H<N>[.W] DestReg, SourceRegStart, SourceRegEnd, [ProfileIdReg,] SourceSizeBytes, HashSizeBits

Instruction Operation: Calculate the selected HASH function on the source and store the result in the DestReg.

- **ProfileIdReg[2:0]:** HASH profile-ID.
  This operand is optional, and relevant to wide-hash operation only
  (i.e. must not be used in short mode).
- Source key is specified in one or more registers, and can be a maximum of 64 Bytes.

• Short mode hash:	Key≤16B
• **.W:** Wide mode hash:	64B≥Key>0B, executed asynchronously in multiple cycles.
❍ For Key ≤ 16B:	Padded by 0's (padding is in the MSBits).
• **.H<N>:**	Hash function selection (relevant to wide-mode only) Short mode: N= 0; Wide mode: N= 0 to 4.
• **HashSizeBits**:	Value: 1 to 32; defines the final hash result size.
• **DestReg**:	4B register. Always cleared before the operation.

! `StartReg= Ri; EndReg= Rj;` where i ≤ j
When the hash input spans multiple registers, the data placement is as in the diagram. This order follows the big-endianess of the MAP registers.



Timing Characteristics:

- **Short mode:** Single Cycle
- **Wide mode:** Multiple cycles, asynchronous operation. The program must ensure the operation is complete by *implicit* SYNC, i.e. by reading the destination register.

Source registers must not be modified until after the operation has completed.

4.13.20 LKP, LKPLPM, LKPT, LKPTI

Function: Table Lookup.

Syntax

LKP:
```
LKP.LF<N>.<R|S|RS>  RdS[.<4Bsel>], RdE[.<4Bsel>], StructID,
AdditionalOffset, RsS[.<4Bsel>], RsE[.<4Bsel>], TableID, KeySize,
KeySizeGranularity, ResultSizeBytes
```

LKPLPM:
```
LKPLPM.LF<N>.<R|S|RS>[.HBH]  RdS[.<4Bsel>], RdE[.<4Bsel>],
StructID, AdditionalOffset, RsS, RsE, TableID, ResultSizeBytes
```

LKPT:
```
LKPT.LF<N>.<R|S|RS>[.PF]  RdS[.<4Bsel>], RdE[.<4Bsel>], StructID,
AdditionalOffset, RsS, RsE, KeySize, ResultSizeBytes, Rm.<4Bsel>
```

LKPTI:
```
LKPTI.LF<N>.<R|S|RS>[.PF]  RdS[.<4Bsel>], RdE[.<4Bsel>], StructID,
AdditionalOffset, RsS, RsE, TableIDImm, KeySize, ResultSizeBytes[,
Rm.<4Bsel>]
```

Instruction Format

LKP: LKP.lookup-result-flag. dest-select DestStart, DestEnd, Struct-ID, AdditionalOffset, SourceStart, SourceEnd, TableID, KeySize, KeySizeGranularity, ResultSizebytes

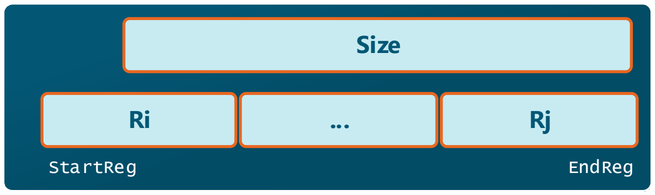
Instruction Operation: Start asynchronous table lookup, store result according to dest-select (either in destination register(s) or struct in context memory (PMEM), or in both), and update the lookup-result-flag with the result.

- **DestStart, DestEnd:** destination register(s)
  - ResultSize≤ 4B:
    Specify a single 4B register (identical for both DestStart and DestEnd).
  - 4B<ResultSize≤16B:
    Specify a single 16B register (identical for both DestStart and DestEnd).
    The specified destination register(s) (either 4B, 16B, or range of 16B registers) is/are cleared before the result is stored.
  - For lookup failures: Failure code is written to **DestEnd[2:0]**
- **SourceStart, SourceEnd:** source register(s) containing the search key.
  - For 4B register: SourceStart and SourceEnd must be identical.
  - For 4B<SearchKey≤16B: Specify a single 16B register.
    (identical for both SourceStart and SourceEnd).

! The data layout for multiple registers is as in the following example:

```
StartReg = Ri; EndReg = Rj; where i ≤ j
```

This order follows the big-endianess of the MAP registers.



- **TableID:** Table identifier
- **KeySizeGranularity: 0:** bits; **1:** bytes
  Bit granularity is used in tables of type ***Direct***, byte granularity is used for all other table types
- **KeySize:** granularity is defined by **KeySizeGranularity**:
  - For Hash Tables: Units: Bytes.
  - For Direct Tables: Units: Bits.
- **ResultSizeBytes:** The size in bytes of the table entry's value field, up to a max of 128B.

  - For a 128B size: *dest-select* must be a Struct.
  - **ResultSizeBytes** *can be **less than*** the entry's value size.
  - If the returned entry's value is larger than ResultSizeBytes, only ResultsSizeBytes (LSBytes) are written back to the destination register(s) or structure.

- **Struct-ID, AdditionalOffset:** The present-bit field in STR.PRESENT, representing where the search result value is to be stored in context memory (PMEM). The exact location in the struct is calculated by adding the additional offset in bytes to the corresponding STR.OFFSET[StructID].
  - **StructID**: Range: 0 to 13.
  - **AdditionalOffset**: Range: 0 to 127.
- **Lookup-result-flag**: 0 to 7 will be set to 1 or 0 according to lookup result.
- **Dest-select:** Select the destination to store the result:
  - **.R:** Store result in `RdS`, `RdE` (StructID is don't care).
  - **.S:** Store result in Struct (`RdS`, `RdE`: Don't-care; RN must be used as `RdS`, `RdE`).
  - **.RS:** Store results in both `RdS`, `RdS` and in Struct.

Instruction Format

LPM: LKPLPM.lookup-result-flag.dest-select DestStart, DestEnd, Struct-ID, AdditionalOffset, SourceStart, SourceEnd, Table-ID, ResultSizebytes

Instruction Operation: Start an LPM table lookup.

Operands Descriptions

- **Lookup-result-flag**: 0 to 7, set to 1 or 0 according to lookup result
- **Dest-select:** Select storage destination for result:
  - **.R**: Result is stored in RdS, RdE (StructID is don't care)
  - **.S**: Result is stored in Struct (RdS, RdE are don't care, RN must be used as RdS, RdE). **.RS**: Result is stored both in RdS and in Struct
- **DestStart**, **DestEnd**: Destination register(s)

  For ResultSize≤ 4B: Specify a single 4B register (identical for DestStart and DestEnd).

  For 4B<ResultSize≤16B: Specify a single 16B register (identical for both DestStart and DestEnd).

  The specified destination register(s) (either 4B, 16B, or range of 16B registers) is/are cleared before the result is stored.

  Lookup failure codes are written to **DestEnd**[2:0]
- **SourceStart**, **SourceEnd**: Source register(s) containing the search key.
- **Search key**: {VRF, IP address} 6B for IPv4; 18B for IPv6.
  - IPv4 key: Specify the key in a single 16B register (SourceStart identical to SourceEnd).
    - **SourceStart**[47:44]: 0 (Zero padding).
    - **SourceStart**[43:32]: VRF.
    - **SourceStart**[31:0]: IP address.
  - IPv6 key:
    - **SourceStart**[15:12]: 0 (Zero padding).
    - **SourceStart**[11:0]: VRF.
    - **SourceEnd**[127:0]: IP address.
- **Struct-ID**, **AdditionalOffset**: The present-bit field in STR.PRESENT represents where the search result value is to be stored in context memory (PMEM).
  The exact location in the STRUCT is calculated by adding the additional offset in bytes to the corresponding STR.OFFSET[StructID].
  - **StructID**: Range: 0 to 13.
- **AdditionalOffset**: Range: 0 to 127.
- **ResultSizeBytes**: Size of table entry's value field, up to a max of 128B. Units: Bytes.

Notes:

- When size is 128B, *dest-select* must be a STRUCT.
- For entry value size ≤ 32B,
  ResultSizeBytes can be less than the entry's value size.
- For entry value size > 32B,
  ResultSizeBytes must be equal to the entry's value size.
- For Returned entry value > **ResultSizeBytes**,
  only ResultsSizeBytes (LSBytes) are written back to the destination register(s) or structure.

Instruction Format

LKPT: `LKPT.lookup-result-flag.dest-select[.PreFwd|.Direct] DestStart, DestEnd,Struct-ID, AdditionalOffset, SourceStart, SourceEnd, KeySize,ResultSizebytes, MetaDataReg`

LKPTI: `LKPT.lookup-result-flag.dest-select[.PreFwd] DestStart, DestEnd, Struct-ID, AdditionalOffset, SourceStart, SourceEnd, TableIDImm, KeySize, ResultSizebytes, MetaDataReg`

Instruction Operation: Start a lookup in TCAM table.

There are two types of TCAM operations: Standard TCAM and Pre-Forwading TCAM:

Standard TCAM

By default, the lookup is performed in the TCAM blocks. A master key of up to 64B is conveyed in [SourceStart, SourceEnd] registers to the selected TCAM instance.

- **MetaDataReg[12:8]**: Descriptor ID specifying the TCAM and master key descriptor to apply.
- For **LKPTI** instruction: Descriptor ID is specified by the **TableIDImm** operand.
- **LFLAG**: Returns operation status: **Range:** NO_MATCH, ANY_MATCH.
- **MetaDataReg**[7:4]: 4-bit field, one bit per lookup operation Bit set to 1 for successful lookup.

  Results are returned to the MAP and placed in [DestStart, DestEnd] registers.

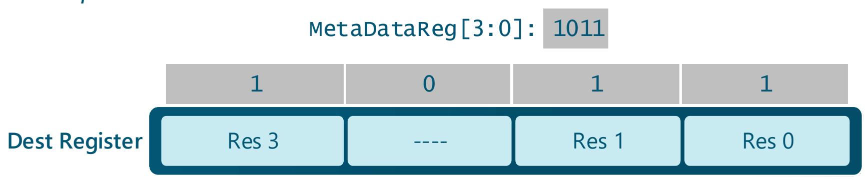
> When writing the status to MetaDataReg[7:4], all unused bits are cleared to 0. (MetaDataReg[31:8] = 0x0).

- **ResultSizeBytes**: Define size of a single result: **0** (4B); **1** (8B).
- **DestStart**, **DestEnd**: Result data written back to these registers.

  The data placement of the results depends on the result size and the selected lookup operations in MetaDataReg[3:0]:

  - Result size 4B:
    - When a single lookup is enabled, a 4B register (`Ri.<4Bsel>`) can be used as DestStart and DestEnd.
    - When multiple lookup operations are enabled, use a single 16B register as DestStart and DestEnd. The placement corresponds to the selection bitmap.

      Example:



- Result size 8B:
  - The user must provide a full 16B register as DestStart and DestEnd. In some cases, either a single register or 2 registers can be provided, and in some cases 2 registers must be provided.
  - Table 7 describes the placement of the results in `{DestStart, DestEnd}` according to the number of enabled lookup operations and the allowed number of result registers:

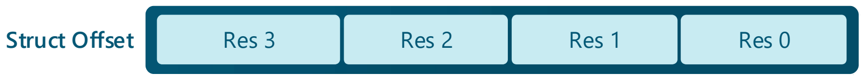
Table 7: Dest Start/End

MetaDataReg[3:0]	DestStart = DestEnd	DestEnd = DestStart+1
0001/0010/0011	[Result1, Result0]	[Result3, Result2, Result1, Result0]
0100/1000/1100	[Result3, Result2]	[Result3, Result2, Result1, Result0]
0111/1011/1101/1110/1111/0110/1001/0101/1010	N/A	[Result3, Result2, Result1, Result0]

For each of the 4 possible lookup operations, if the lookup is disabled or NO_MATCH response, zero is written to the corresponding result.

- Result write-back to a struct:

  When the lookup results are targeted to a struct, all 4 results (or 0) are always written, regardless of the number of lookup operations:



  - For Result Size=4B: 16B are written.
  - For Result Size=8B: 32B are written.

Pre-Forwading TCAM:

- When **.PF** is opted, the lookup is performed on the pre-forwarding TCAM. This TCAM supports a single lookup operation.
  - **MetaDataReg**[10:8]: The PF-TCAM master key descriptor ID to apply for the lookup operation.

    For **LKPTI** instruction, this is specified by **TableIDImm**.
  - **MetaDataReg**[4]: Lookup operation succeeded (**1**) or failed (**0**).

    > When writing the status to MetaDataReg[4], all other bits are cleared (MetaDataReg[3:0]=0 and MetaDataReg[31:5]=0.

- *LFLAG* option: Returns operation status: **MATCH** or **NO_MATCH**.
- **ResultSizeBytes:** Defines the size of a single result: **0** (4B); **1** (6B).
- **DestStart**, **DestEnd**: Must be the same register - size depends on the result size (4B for a 4B Result Size, 16B for a 6B Result Size).
  The result data is written back to this register. The data placement of the result depends on the result size.
  - Result size 4B: The full 32b of the 4B register are written back.
  - Result size 6B: The 48 LSbits (**DestEnd[47:0]**) are written back.

Errors

- When a **TCAM** lookup failure occurs, an error code may be returned in DestEnd[2:0]
  - **No error code** is returned when at least one of multiple lookups succeeded.
    The results are reflected in **MetaDataReg**[7:4] as explained above.
- When a **PF-TCAM** lookup failure occurs, an error code is returned in **DestEnd[2:0]**
  - **SourceStart, SourceEnd**: Source register(s) containing the search key.

    If the search key is 16B, specify a single 16B register (identical for both SourceStart and SourceEnd).
  - **Struct-ID, AdditionalOffset**: The present-bit field in STR.PRESENT, represents where the search result value is to be stored in context memory (PMEM). The exact location in the STRUCT is calculated by adding the additional offset in bytes to the corresponding STR.OFFSET[**StructID**].

■ **StructID**	**Range:** 0 to 13.
■ **AdditionalOffset**	**Range:** 0 to 127.

  - **KeySize**: N×16B   N = 1..4;   i.e. `KeySize`=16B/32B/48B/64B.

4.13.21 SYNC, SYNCALL

Function: Wait for the completion of pending asynchronous operation (e.g. lookup, AQM or wide-Hash operation, ST, etc.)

Syntax

SYNC: `SYNC[.N] Bitmap, [PC]`

SYNCALL: `SYNCALL[.N] Bitmap, [PC]`

Instruction Format

SYNC: SYNC[.N] FlagsBitmap, [PC]

Instruction Operation: Sync on specific asynchronous operations specified by FlagsBitmap.

If PC is specified, it is the jump-to address upon match/no match, when **.N:**

- ***is opted***, PC denotes the jump address upon *no* match
  (of at least one of the indicated LFLAGs).
- ***is not opted***, PC denotes the jump address upon match (of all the indicated LFLAGs).

> - PC is optional and is relevant only if the FlagsBitmap value is not zero
> - This option is relevant only when a lookup action that has a match/no-match result is used by one of the LFLAGs.
>   Normally, only a single such action is synchronized by this instruction.
> - When multiple LFLAGS are specified in the bitmap, the jump is taken after all operations completed.

Instruction Format

SYNCALL: SYNCALL[.N] FlagsBitmap, [PC]

Instruction Operation: Sync on lookup operations denoted by FlagsBitmap, and for all other asynchronous operations (e.g. MOD, HASH, AQM, etc.)
If PC is specified, it is the jump-to address upon match/no match, when **.N:**

- ***is opted***, PC denotes the jump address upon *no* match.
- ***is not opted***, PC denotes the jump address upon match.

> - PC is optional and is relevant only if the FlagsBitmap value is not zero.
> - This option is relevant only when a lookup action that has a match/no-match result is used by one of the LFLAGs.
>   Normally, only a single such action is synchronized by this instruction.
> - When multiple LFLAGS are specified in the bitmap, the jump is taken after all operations are completed.

Timing Characteristics: Multiple cycles, stalled until all requested asynchronous operations are completed.

4.13.22 HALT

Function: Ends the MAP thread execution.

Syntax

```
HALT
```

Instruction Format

HALT

Instruction Operation: Halt the MAP program.

Timing Characteristics: Single Cycle

4.13.23 CPR, CP, CPI, CPS, CPIS, CPH, CPIH, CPF

Function: Copy data to packet header.

Syntax

CPR:	`CPR.LF<N> SignedOffsetSizeBytesReg.<4Bsel>, RsS, RsE`
CP:	`CP.LF<N> SignedOffsetReg.<4Bsel>, RsS, RsE, SizeBytes`
CPI:	`CPI.LF<N> SignedOffsetImm, RsS, RsE, SizeBytes`
CPS:	`CPS.LF<N> SignedOffsetReg.<4Bsel>, StructID, AdditionalOffsetImm, SizeBytes`
CPIS:	`CPIS.LF<N> SignedOffsetImm, StructID, AdditionalOffsetImm, SizeBytes`
CPH:	`CPH.LF<N> SignedOffsetReg.<4Bsel>, HdrOffsetID, AdditionalOffsetImm, SizeBytesReg.<4Bsel>`
CPIH:	`CPIH.LF<N> SignedOffsetImm, HdrOffsetID, AdditionalOffsetImm, SizeBytesImm`
CPF:	`CPF.LF<N> Rs.<4Bsel>`

Instruction Format

CPR: CPR.LF<N> SignedOffsetSizeBytesReg, SourceRegStart, SourceRegEnd

CP: CP.LF<N> SignedOffsetReg, SourceRegStart, SourceRegEnd, SizeBytesImm

CPI: CPI.LF<N> SignedOffsetImm, SourceRegStart, SourceRegEnd, SizeBytesImm

Instruction Operation: Copy data from source registers to packet header position calculated by cursor (frame offset) + SignedOffset.

- **SizeBytes:** Immediate value. **Range:** 1 to 128.
- **SignedOffset:** **Range:** -224 to +255 (9b), limited to the actual packet header size.
- **SignedOffsetSizeBytesReg[8:0]:** The signed offset in the header to copy to.
- **SignedOffsetSizeBytesReg[23:16]:** Copy size. **Units:** Bytes.

Instruction Format

CPS: CPS.LF<N> SignedOffsetReg, StructID, AdditionalOffsetImm, SizeBytes

CPIS: CPIS.LF<N> SignedOffsetImm, StructID, AdditionalOffsetImm, SizeBytes

Instruction Operation: Copy data from Struct to packet header.

Source data indicated by STR.OFFSET[StructID]+ AdditionalOffset.

Copy location in packet header is calculated by cursor (frame offset) + SignedOffset.

- **SignedOffset:** Range: -224 to +255 (9b), limited to the actual packet header size.

> - StructID is always assumed to be set.

Instruction Format

CPH: CPH.LF<N> SignedOffsetReg.<4Bsel>, HdrOffsetID, AdditionalOffsetImm, SizeBytesReg

CPIH: CPIH.LF<N> SignedOffsetImm, HdrOffsetID, AdditionalOffsetImm, SizeBytesImm

Instruction Operation: Copy data from packet header to packet header (copy data in place). The source data is indicated by HdrOffset+ AdditionalOffset, and position in the packet header is calculated by cursor (frame offset) + SignedOffset.

- **SizeBytesReg[7:0]** **Range:** 1 to 128.
- **SignedOffset** **Range:** -224 to +255 (9b).
  Limited to the actual packet header size.

Instruction Format

CPF: CPF.LF<N> SrcReg

Instruction Operation: Copy data from the current packet (frame) header to allocated linked buffers.

- **SrcReg** **Size:** 4B register.
  - **SrcReg[8:0]:** Offset in header to copy from. **Range**: -224 to +255 (9b); limited to actual packet header size.
  - **SrcReg[16:9]:** Offset in first buffer to copy to. **Range**: 0 to 255.
  - **SrcReg[27:20]:** Size bytes to copy. **Range**: 1 to 256; Use value **0** for 256B.
  - **SrcReg[30:28]:** HW index of first buffer to copy to. **Range**: First ***or*** Second buffer. No need for subsequent buffer indices.

Timing Characteristics:

Asynchronous operation.
Program must perform **SYNC** on LFLAG to ensure operation is completed.

4.13.24 CHKSUMTST, CHKSUMUPD, CHKSUMCALC

Function:

CHKSUMTST:	Check correctness of IPv4 header checksum.
CHKSUMUPD:	Calculate and update checksum in IPv4 header.
CHKSUMCALC:	Calculate checksum of a variable portion of the packet header.

Syntax

CHKSUMTST: `CHKSUMTST.LF<N> HeaderOffsetID`

CHKSUMUPD: `CHKSUMUPD.LF<N> HeaderOffsetID`

CHKSUMCALC: `CHKSUMCALC.LF<N> Rd.<4Bsel>, HeaderOffsetID, AdditionalOffset, Rs.<4Bsel>`

Instruction Format

CHKSUMTST: CHKSUMTST.LF<N> HeaderOffsetID

Instruction Operation: Calculates IPv4 checksum in the packet header, and sets LFLAG to true/false if result is correct/incorrect. The IPv4 header location is indicated by HeaderOffsetID.

The operation is asynchronous, result is valid after performing **SYNC** on the selected LFLAG (**.LF<N>**).

Instruction Format

CHKSUMUPD: CHKSUMUPD.LF<N> HeaderOffsetID

Instruction Operation: Calculates the IPv4 header checksum and writes back the result to the packet header. The IPv4 header location is indicated by HeaderOffsetID.

The operation is asynchronous, result is valid after performing **SYNC** on the selected LFLAG (**.LF<N>**).

Instruction Format

CHKSMCALC: CHKSUMCALC.LF<N> DestReg, HeaderOffsetID, AdditionalOffset, SizeReg

Instruction Operation: DestReg[15:0] = 16b checksum, calculated from packet header location indicated by HeaderOffsetID + AdditionalOffset, on the number of bytes indicated by SizeReg.

SizeReg[7:0]:	Data size. **Range:** 1 to 128. Units: 2B

Since the operation is asynchronous, the result is valid after performing **SYNC** on the selected LFLAG (**.LF<N>**).

Timing Characteristics: Asynchronous operation

4.13.25 SENDOUT, SENDOUTI, SENDQID, SENDDATA, SENDDATAI, DROP

Function: Send packet (frame) to external port, or drop the packet, and optionally halt.

Syntax

SENDOUT: `SENDOUT[.LF<N>.CLONE|MIRR|H] Rs1, Rs2[.<4Bsel>], BufferDeltaImm`

SENDOUTI: `SENDOUTI[.LF<N>.CLONE|MIRR|H] Rs1, Rs2[.<4Bsel>], FrameDeltaSizeImm, BufferDeltaImm`

SENDQID: `SENDQID.LF<N>[.CLONE|MIRR] Rs1, BufferDeltaImm`

SENDDATA: `SENDDATA[.LF<N>.CLONE|MIRR|.H] Rs1, Rs2[.<4Bsel>], BufferDeltaImm`

SENDDATAI: `SENDDATAI[.LF<N>.CLONE|MIRR|H] Rs1, Rs2[.<4Bsel>], FrameDeltaSizeImm, BuffersDeltaImm`

DROP: `DROP[.LF<N>.CLONE|MIRR][.H] BufferDeltaImm`

The **SENDQID** and **SENDDATA** instructions are a breakdown of the **SENDOUT** operation into two steps, thus enabling improved performance in a typical program flow.

The improved performance is achieved by allowing the program to enqueue the packet while packet editing is still on-going.

Instruction Format

SENDOUT: SENDOUT[.LF<N>.CLONE|MIRR|H], ParamsReg, MACE_DataReg, BufferDeltaImm

SENDOUTI: SENDOUTI[.LF<N>.CLONE|MIRR|H], ParamsReg, MACE_DataReg, FrameDeltaImm, BufferDeltaImm

SENDQID: SENDQID.LF<N>[.CLONE|MIRR], ParamsReg, BufferDeltaImm

SENDDATA: SENDDATA[.LF<N>.CLONE|MIRR|H], ParamsReg, MACE_DataReg, BufferDeltaImm

SENDDATAI: SENDDATAI[.LF<N>.CLONE|MIRR|H], ParamsReg, MACE_DataReg, FrameDeltaImm, BufferDeltaImm

Instruction Operation: Send the packet to an external port, with MAC editor command and data.

- **.LF<N>:** This option is mandatory unless **.H** is specified.
- Since this is an asynchronous operation, for mirroring, the program must perform a **SYNC** on the selected LFLAG (**.LF<N>**) before attempting to rebase the header pointers (via **FREBASE**).
- **SENDQID** operation can be explicitly SYNC'ed by its LFLAG, and is also implicitly SYNC'ed by **SENDDATA**.

ParamsReg content is different for each instruction, as follows:

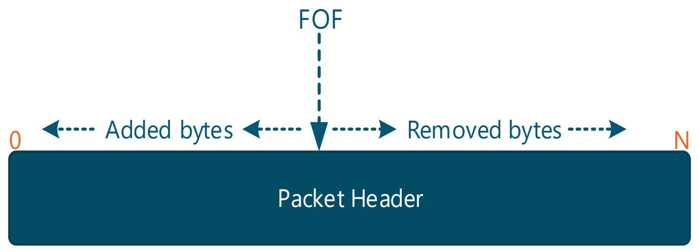
- **SENDQID:** SizeHint, IAI, PFC_PRIO, Q_CNG, SkipFlitsDisable, TargetQID
- **SENDOUT** SizeHint, FrameDelta, IAI, PFC_PRIO, Q_CNG, SkipFlitsDisable, TargetQID
- **SENDDATA:** SizeHint, FrameDelta, IAI, PFC_PRIO, Q_CNG, SkipFlitsDisable, TargetQID
- **SENDOUTI:** Same as **SENDQID** (**FrameDeltaImm** is a separate operand).
- **SENDDATAI**:Same as **SENDQID** (**FrameDeltaImm** is a separate operand).

ParamsReg structure:

- ParamsReg[15:0]: Target Queue ID **(TargetQID)**
- ParamsReg[16:16]: Queue-Congestion Indication (**Q_CNG)** conveyed to the target queue.
- ParamsReg[24:24]: Ingress Accounting Indication (**IAI**).
- ParamsReg[27:25]: PFC priority for ingress accounting (**PFC_PRIO**)
  Note: Relevant only when **IAI** is set.
- ParamsReg[40:32]: **FrameDelta**: See below.
- ParamsReg[41:41]: **SkipFiltsDisable**:The MAP code must set this bit when the packet is sent to a multicast (loopback) queue, and clear this bit otherwise.
- **BufferDeltaImm**: Specifies the additional number of buffers.
  Range: 0, 1.

FrameDelta The number of bytes added/removed to the header, relative to the current frame offset (FOF).

> The number of bytes added is indicated by a positive value.
>
> Removed bytes are indicated by a negative value.



- **ParamsReg[40:32]:** The 9b value is taken as signed integer.
- When the packet is transmitted, the start of packet is calculated as

  FOF - FrameDelta

- **MAC-Editing**: The MACE vector and related data fields.
- **.H:** Halt the program.

- **.CLONE:** This option is used when a packet is cloned for mirroring. This option should be used when SENDOUT/SENDIQ is called for the original packet, which should always be called before sending the mirrored packet.

  > For mirrored packets, when SENDOUT.CLONE is executed, then all packet buffers with ref-count = 0 are released only when both the original packet and the mirrored packet are sent or dropped. It is, therefore, important that all conditions for sending/dropping the mirrored packet are met before SENDOUT.CLONE is executed.

- **.MIRR:**
  - ❍ Use this instruction to send the mirrored packet to the destination queue.

    SENDQID.MIRR can be used as the first step, only when followed by **SENDDATA.MIRR**
  - ❍ **.LF<N>** is required, but **SYNC** for this LFLAG is not required.
  - ❍ **.H** can be used with **.MIRR**
- **BufferDeltaImm:** Specifies the additional number of buffers. Range: 0 or 1.

Function: **DROP:** DROP[.LF<N>.CLONE|MIRR][.H] BufferDelta

Instruction Operation: Drop the packet.

- **.LF<N>:** This option is required only with **.CLONE** or **.MIRR** options, and must not be used otherwise. The operation is asynchronous, and in mirroring case, the program must perform **SYNC** on the selected LFLAG (**.LF<N>**) before attempting to rebase the header pointers (via **FREBASE**).
- **.CLONE:** This option is used when a packet is cloned for mirroring, however, the original packet is dropped.

  The flow is similar to when **SENDOUT.CLONE** is used, with the exception that the original packet is dropped.

> For a mirrored packet, when DROP.CLONE is executed, then all packet buffers with ref-count = 0 are released only when both the original packet and mirrored packet are sent or dropped. It is, therefore, important that all conditions for sending/dropping the mirrored packet are met before DROP.CLONE is executed.

- **.MIRR:** Select this option when a mirrored packet is dropped, after executing **SENDOUT/QID/DATA/DROP.CLONE**.
- **BufferDelta** Specifies the number of additional buffers in the *local packet header*. **Range:** 0 or 1.
- **.H:** Halt the program.
- **.CLONE** and **.MIRR** are mutually exclusive, as well as **.CLONE** and **.H**.
- **DROP.MIRR.H** is supported.

Timing Characteristics: **Asynchronous operation**

4.13.26 COUNTER

Function: Increment/decrement/set-and-read a counter vector.

Syntax

COUNTER: `COUNTER[.NP][.DECR|.SET] Rd[.<4Bsel>], Rs1[.<4Bsel>], Rs2.<4Bsel>, TableIdImm, CounterSizeImm, VectorSizeImm`

Instruction Format

COUNTER: COUNTER[.NonP][.Decr|.Set] DestReg, SrcReg1, SrcReg2, TableIdImm, CounterSizeImm, VectorSizeImm

Instruction Operation: Perform an atomic counter operation:

- **Increment (Default):** Add the value(s) in SrcReg1 to the counter index in SrcReg2.
  In non-posted operations, the new counter value(s) are returned in DestReg.
- **Decrement** (**.Decr** option)**:** Subtract the value(s) in SrcReg1 from the counter index in SrcReg2.
  In non-posted operations, the new counter value(s) are returned in DestReg.
- **Set** (**.Set** option): Overwrite the counter value(s) with the values in SrcReg1 from the counter index in SrcReg2.
  In non-posted operation, the counter value(s) ***prior*** to overwriting are returned in DestReg.
- **.NonP:** Non-posted operation.
  Default: Posted operation.

> When non-posted is opted, **DestReg** must be a valid register.
> For posted operations, **DestReg** must be a null register.

- **SrcReg1:** Holds the delta value of the counter set, at bit offset 0.
  Example: Counter set vector size is 2, and counter size is 4B:
  The counter set delta is: {SrcReg1[63:32], SrcReg1[31:0]}
- **SrcReg2[17:0]:** Holds the counter set index in the table.
- **TableIdImm:** Holds the table descriptor ID.
- **CounterSizeImm:** **Range:** 4, 8. Units: Bytes.
- **VectorSizeImm:** Number of counters in the vector. **Range:** 1, 2.

Timing Characteristics: Single Cycle, Asynchronous operation.

For non-posted operations, the result is returned asynchronously, a pseudo-**SYNC** is performed implicitly when reading the DestReg.

4.13.27 METER

Function: Activate a meter (traffic policer) operation.

Syntax

METER: `METER Rd.<4Bsel>, Rs1.<4Bsel>, Rs2.<4Bsel>, TableIdImm`

Instruction Format

METER: METER DestReg, SrcReg1, SrcReg2, TableIdImm

Instruction Operation: Activate a meter and get the resulting color.

- **DestReg[1:0]:** Holds the result of the meter operation, the conformance level
  Range:
  0: Green;
  1: Yellow;
  2: Red.
- **SrcReg1[1:0]:** Holds the ingress color of the packet (0: G, 1: Y, 2: R).
- **SrcReg1[25:2]:** Holds the charged length.
- **SrcReg2[17:0]:** Holds the meter index in the table.

  The actual index size in bits is determined by the meter table size (taken from table descriptor).
- **TableIdImm:** Holds the table descriptor ID.

Timing Characteristics: Single Cycle, Asynchronous operation.
The result is returned asynchronously, a pseudo-**SYNC** is performed implicitly when accessing the DestReg.

4.13.28 CAS, TAS

Function:

CAS: Compare and swap value in RAM as an atomic operation.

TAS: Test and set a bit value in RAM as an atomic operation.

Syntax

CAScc:
```
CAS<condition-code>[.NP][.ABS] Rd[.<4Bsel>], Rs1.<4Bsel>,
Rs2[.<4Bsel>], RecordSizeImm
```

TAScc:
```
TAS<condition-code>[.NP][.ABS] Rd.<4Bsel>, Rs1.<4Bsel>,
Rs2.<4Bsel>, BitOffsetImm
```

Instruction Format

CAS: CAS<*condition-code*>[.NonP][.ABS] DestReg, SrcReg1, SrcReg2, RecordSizeImm

Instruction Operation: Perform an atomic compare-and-swap operation in RAM, or activate a Packet Descriptor.

- **SrcReg1[31:0]:** RAM address on which to perform the operation.
- **CAS<condition-code>:**
  - **Vin**: Data value read from the RAM address.
  - **T**: Threshold value from SrcReg2.
  - **Vout**: Data value from SrcReg2.
  - **S**: Condition-code met and operation was executed.
  - **<condition-code>:** If comparison condition (below) is true, write **Vout** to RAM location, and set **S** to true:
    - **EQ**: **Vin == T**
    - **NEQ**: **Vin != T**
    - **LE**: **Vin ≤ T**
    - **GE**: **Vin ≥ T**

> Condition codes `LE`/`GE` are not supported for record size 8B.

- **.NonP:** Non-posted operation.
  Default: Posted operation.

  When **.NonP** is opted, **Vin, S** are returned to **DestReg:**
  - **DestReg[0:0]:** **S** value.
    1: Operation performed;
    0: Operation was not performed.
  - **DestReg[N+1:1]:** **Vin** value: The record content before the operation.
  - **N:** RecordSizeImm × 8

> When non-posted is opted, **DestReg** must be a valid register.
> For posted operations, **DestReg** must be a null register.

- **.ABS:** The RAM address provided in **SrcReg1** is absolute.
  When **.ABS** is not opted, the RAM address provided in **SrcReg1** is in a local MBR.

> Address is always 4B aligned unless record-size is 8, in which case address is 8B aligned.

- **RecordSizeImm:** RAM data size. Units: Bytes. Range: 1, 2, 4, or 8.
  If record size is 1 or 2, the record must be 4B aligned.
- **SrcReg2:** Holds the threshold value for the comparison and the data to write to RAM if the comparison condition is met.
  - **SrcReg2[N-1: 0]:** The data to write (**Vout**).
  - **SrcReg2[N×2 -1: N]:** The threshold for the comparison
    - **N:** RecordSizeImm × 8

Instruction Format

TAS: TAS<condition-code>[.NonP][.ABS] DestReg, SrcReg1, SrcReg2, BitOffsetImm

Instruction Operation: Perform an atomic test-and-set bit operation in RAM location of a 16-bit value.

- **TAS<condition-code>:**
  - **Vin**: A 16bit value read from RAM location (address specified in **SrcReg1**).
  - **T**: Bit value from **SrcReg2**.
  - **Vout**: Data value from **SrcReg2**.
  - **<condition-code>:** If comparison condition (below) is True, write **Vout** to bit location in RAM:
    - **EQ: Vin[BitOffsetImm] == T**
    - **NEQ: Vin[BitOffsetImm] != T**
- **.NonP:** Non-posted operation.
  Default is posted operation.

  If **.NonP** is opted, **DestReg[0]** is set to **1** if the **Vout** bit was written;
  otherwise, **DestReg[0]** is cleared to **0**.

> When non-posted is opted, **DestReg** must be a valid register.
> For posted operations, **DestReg** must be a null register.

- **SrcReg1[31:0]:** An address of a 16b value in RAM (2B-aligned).
- **.ABS:** The RAM address provided in **SrcReg1** is absolute.
  When **.ABS** is not opted, the RAM address provided in **SrcReg1** is in local RAM.
- **BitOffsetImm:** The bit location (0-15) in RAM address (specified by **SrcReg1**) of a 16-bit value.
- **SrcReg2:** Holds the threshold bit value for the comparison and the data bit value to write to RAM in case the comparison condition is met.
  - **SrcReg2[0:0]:** The data to write.
  - **SrcReg2[1:1]:** The threshold for the comparison.

Timing Characteristics: Single Cycle, Asynchronous operation. For non-posted operations, the result is returned asynchronously, a pseudo-**SYNC** is performed implicitly when accessing the DestReg.

4.13.29 BWAND, BWOR, BWXOR, BWSHR, BWSHL

Function: Bitwise operation on RAM as atomic operation.

Syntax

BW:
```
BW[operation][.NP][.ABS] Rd[.<4Bsel>], Rs1.<4Bsel>, Rs2.<4Bsel>,
RecordSizeImm
```

Instruction Format

BW: BW[operation][.NonP][.ABS] DestReg, SrcReg1, SrcReg2, RecordSizeImm

Instruction Operation:

- **BW[operation]:** Perform the bitwise operation.
- **SrcReg1[31:0]**: RAM address on which to perform the operation.
- **Vin:** Data value read from the RAM location specified by **SrcReg1**, of size **RecordSizeImm**.
  - **Data**: Data value from **SrcReg2**.
  - **Vout = Vin *operation* Data**

    Operation: AND, OR, XOR, SHL or SHR.
  - **Vout is written back to the RAM location.**
- **.NonP:** Non-posted operation. Default is posted operation.
  If **.NonP** is opted, **Vout** is returned to **DestReg**.

> When non-posted is opted, **DestReg** must be a valid register.
>
> For posted operations, **DestReg** must be a null register.

- **.ABS:** The RAM address provided in **SrcReg1** is absolute.
  When **.ABS** is not opted, the RAM address provided in **SrcReg1** is in local RAM.
- **RecordSizeImm:** The size of the RAM data in bytes: 1/2/4.

Timing Characteristics: Single Cycle, Asynchronous operation.

For non-posted operation, the result is returned asynchronously, a pseudo-**SYNC** is performed implicitly when accessing the DestReg.

4.13.30 DLB

Function: Dynamic load balancing atomic operation.

Syntax

DLB: `DLB Rd.<4Bsel>, Rs1.<4Bsel>, Rs2.<4Bsel>, TableIdImm`

Instruction Format

DLB DestReg, SrcReg1, SrcReg2, TableIdImm

Instruction Operation: Perform DLB (dynamic-load balancing) operation on a data flow. The program suggests a new destination if the packet inter-arrival time threshold is exceeded. The response offers either the previous destination or the new destination, depending on the time passed since previous packet on this flow.

• **TableIdImm**	The DLB table-ID.
• **Src1[18:0]**	The minimal inter-arrival time for the flow. Units: microseconds.
• **Src1[30:20]**	Suggested new destination.
• **Src2[17:0]**	The DLB record index.
• **DestReg[10:0]**	The offered destination.

Timing Characteristics: Asynchronous non-posted operation.

The result is returned asynchronously.
A pseudo-**SYNC** is performed implicitly when accessing the DestReg.

4.13.31 LDRTC

Function: Load RTC content to register.

Syntax

```
LDRTC   Rd
```

Instruction Format

LDRTC: LDRTC DestReg

Instruction Operation: Load the content of RTC to destination register.

- RTC is a 40b value of the real-time clock structured as follows:
  - **RTC[39:29]**: Seconds counter.
  - **RTC[28:0]**: 2 nano-second counter (sub-seconds part).
- The seconds part is read into DestReg as is;
  The sub-seconds part is read as a 1ns counter:
  - **DestReg[0:0] = 0**
  - **DestReg[29:1] = RTC[28:0]**
  - **DestReg[40:30] = RTC[39:29]**

Timing Characteristics: Single Cycle

4.13.32 LDID

Function: Load MAP HW-ID.

Syntax

```
LDID Rd.<4Bsel>
```

Instruction Format

LDID DestReg

Instruction Operation: Load the MAP HW-ID to a destination register.

- **DestReg[13:0]** = HW-ID

Timing Characteristics: Single Cycle

4.13.33 AQMEG, AQMLD

Function: Active Queue Management (AQM) query.

Syntax

AQMEG: `AQMEG.LF<N>[.NOMIRR] Rd.<4Bsel>, Rs.<4Bsel>`

AQMLD: `AQMLD.LF<N> Rd, Rs.<4Bsel>`

Instruction Format

AQMEG: AQMEG.LF<N>[.NOMIRR] DestReg, ParamsReg

Instruction Operation: Perform egress AQM and provide result in DestReg.

Options:

- **.NOMIRR:** Perform AQM without MIRROR evaluation.
  When selected, MIRROR flag in the result is N/A.

Egress parameters:

  - **ParamsReg[15:0]:** Egress QID
  - **ParamsReg[25:24]:** Drop Precedence (DP)

- **DestReg:** Includes the egress AQM result and additional data depending on the selected options:
  - **DestReg[2:0]** include AQM result:
    - Bit [0]: doDROP (1: Yes; 0: No)
    - Bit [1]: Always 0
    - Bit [2]: doMirror (1: Yes; 0: No)
  - **DestReg[3:3]:** Includes the **InBurst** indication for the egress queue (aka *uBurst* detection).

> All N/A fields in DestReg are cleared to 0 by default.

Instruction Format

AQMLD: AQMLD.LF<N> DestReg, ParamsReg

Instruction Operation: Retrieve AQM metadata.

Parameters:

- **ParamsReg[15:0]**: Egress QID
- **ParamsReg[25:24]**: Drop Precedence (DP)
- **DestReg[115:0]**: AQM metadata

Timing Characteristics: Asynchronous operation

4.13.34 RAND

Function: Generate a random number.

Syntax

```
RAND Rd.<4Bsel>
```

Instruction Format

RAND: RAND DestReg

Instruction Operation: Generates a 32b random number and stores it in DestReg. Result is 32b size.

Timing Characteristics: Asynchronous operation.

4.13.35 STALLOC

Function: Allocate a struct in PMEM.

Syntax

```
STALLOC StructID, SizeBytes
```

Instruction Format

STALLOC: STALLOC StructID, SizeBytes

Instruction Operation: Allocate a struct of size SizeBytes in PMEM, set Struct-ID bit in STR.PRESENT, and update the offset in STR.OFFSET byte index StructID.

- A struct in PMEM is always 4-byte aligned. The offset field value in the internal STR.OFFSET register is in 4B granularity.
- **StructID:** Immediate value, 1 to 13.
- **SizeBytes:** Immediate value, 4 to 256 bytes.

Timing Characteristics: Single cycle

4.13.36 STRGET

Function: Get STR.PRESENT and STR.OFFSET internal registers.

Syntax

```
STRGET Rd
```

Instruction Format

STRGET: STRGET DestReg

Instruction Operation: Read STR.PRSENT and STR.OFFSET into DestReg.

DestReg: Full 16B register.

- STR.OFFSET bytes are placed in DestReg bytes 0 to 13 (DestReg[127:16]).
- STR.PRESENT bits are placed in DestReg bytes 14 or 15 (DestReg[13:0]).

Timing Characteristics: Single cycle

4.13.37 STRSET

Function: Set STR.PRSENT and STR.OFFSET internal registers, and optionally move the internal structure allocation cursor position.

Syntax

```
STRSET Rs [, CursorDelta]
```

Instruction Format

STRSET: STRSET SourceReg [, CursorDeltaImm]

Instruction Operation: Write to STR.PRSENT and STR.OFFSET the SourceReg, and optionally update the cursor.

- **SourceReg**: Full 16B register.
  - STR.OFFSET bytes are placed in SourceReg bytes 0 to 13 (SourceReg[127:16]).
  - STR.PRESENT bits are placed in SourceReg bytes 14 or 15 (SourceReg[13:0]).
- **CursorDeltaImm:** Increment/decrement internal cursor;
  Signed value ***in 4B granularity***;
  Range: -64 to +64 (enables moving the cursor ±256 bytes).

Timing Characteristics: Single cycle

4.13.38 STRGETCUR

Function: Get the internal structure allocation cursor.

Syntax

```
STRGETCUR Rd.<4Bsel>
```

Instruction Format

STRGETCUR: STRGETCUR DestReg

Instruction Operation: DestReg[7:0] = Internal structure allocation cursor.

The cursor value is in 4B granularity.

Timing Characteristics: Single cycle

4.13.39 STRSETCUR, STRSETCURI

Function: Set the internal structure allocation cursor.

Syntax

STRSETCUR:`STRSETCUR Rs.<4Bsel>`

STRSETCURI:`STRSETCURI CursorImm`

Instruction Format

STRSETCUR: STRSETCUR SourceReg

STRSETCURI:`STRSETCURI CursorImm`

Instruction Operation: Set the internal structure allocation cursor to a new absolute value.

- The new absolute value of the cursor is in **SourceReg[7:0]**.
- ***The value is in 4B granularity***. Range: 0 to 255.

Timing Characteristics: Single cycle

4.13.40 LBALLOC

Function: Allocate buffers in local RAM and link to packet header buffer.

- **LBALLOC**: Allocate and link new buffers to original buffers.
- FREBASE, FREBASEI: Rebase the header pointer between the packet header buffers, and adjust the FrameOffset cursor and the internal header size.
- GETFPTR: Retrieve the current header buffers' pointers.
- CPF: Copy data from the current header to a list of linked buffers.

Syntax

```
LBALLOC.LF<N> Rd.<4Bsel>, NumBuffersImm, LinkedToBuffImm, RefCnt
```

Instruction Format

LBALLOC.request-flag DestReg, NumBuffersImm, LinkedToBuffImm, RefCount

Instruction Operation: Allocate 1 or 2 **NumBuffersImm** buffers.

If two NumBuffersImm are allocated, then they must be linked together in the following manner:

- Link the last buffer to the packet header buffer **LinkedToBuffImm**, and set the reference-count of the allocated buffer(s) to **RefCount**.
- The instruction returns in **DestReg** the HW pointer(s) of the allocated buffer(s).
  - **NumBuffersImm:** Number of buffers to allocate. Range: 1 or 2.

    The total header size can be increased by a maximum of only 1 buffer.

    When allocating 2 buffers, the buffers are linked to each other and then linked to the 2nd header buffer (i.e. **LinkedToBuffImm** must be 1).
  - **LinkedToBuffImm:** The logical buffer index in the original header to which the new last buffer is linked. Range: 0, 1.
  - **RefCount:** Reference-count value of the allocated buffers. Range: 0, 1.

    > For multicast flow, **RefCount** must always be 1.

  - **DestReg[2:0]:** HW buffer index of the first allocated buffer.
  - **DestReg[18:16]:** HW buffer index of the second allocated buffer

    In case of failure, HW writes-back 0 to **DestReg**.
- The operation is asynchronous, and is finalized by the MAP program by executing **SYNC.N Bitmap, PC**.

Timing Characteristics: Asynchronous operation, implicitly sync'ed when **DestReg** is accessed.

4.13.41 FFLUSH

Function: Flush all packet (frame) header data from cache.

Syntax

```
FFLUSH.LF<N>
```

Instruction Format

FFLUSH.LF<N>

Instruction Operation: Flush any valid packet header data from cache to RAM.

Timing Characteristics: Asynchronous operation, the program must call **SYNC** for the selected LFLAG before proceeding with **LBFREE/REPARSE**.

4.13.42 LBFREE

Function: Free buffers previously allocated by `LBALLOC`.

Syntax

```
LBFREE Rs.<4Bsel>, NumBuffersImm
```

Instruction Format

LBFREE BufferPtrReg, NumBuffersImm

Instruction Operation: Free local buffers previously allocated by `LBALLOC`.

- **BufferPtrReg[2:0]:** HW index of the first buffer to free.
- **NumBuffersImm:** Amount of buffers to free, that are linked to the first buffer. Range: 1, 2.
- The reference-count of the buffers is reset to 0 by HW.

Timing Characteristics: Asynchronous operation

4.13.43 FREBASE, FREBASEI

Function: Re-base packet (frame) header buffer pointers

Syntax

FREBASE: `FREBASE.LF<N>[.SH] Rd.<4Bsel>, Rs1.<4Bsel>, Rs2.<4Bsel>`

FREBASEI: `FREBASEI.LF<N>[.SH] Rd.<4Bsel>, Rs1.<4Bsel>, DeltaHdrSizeImm`

Instruction Format

FREBASE: FREBASE.LF<N>[.SH] DestReg, NewB0Reg, DeltaHdrSizeReg

FREBASEI: FREBASEI.LF<N>[.SH] DestReg, NewB0Reg, DeltaHdrSizeImm

Instruction Operation: Re-base buffer pointers of packet header, and adjust the header size by DeltaHdrSize. The packet frame-offset cursor is also adjusted accordingly.
The instruction returns the previous frame-offset value.

- **NewB0Reg[2:0]:** The HW buffer index to the start of the packet to which the MAP is re-based. This buffer index can be either a previously allocated and linked buffer (via **LBALLOC**), or the original **B0** buffer pointer.
- **DeltaHdrSize[4:0]:** a signed number ***in flit granularity (32B)***, to add to current header-size.

  Range: -8 to +8 (i.e., -256B to +256B in 32B increments), and the frame-offset is adjusted accordingly (in modulu-8).

> Since the FOF value is 0 to 7:
> - A DeltaHdrSize value of `8-FOF`, which brings FOF to 0 (start of next buffer) is illegal.
> - A DeltaHdrSize value `-FOF`, which brings FOF to 0 (start of current buffer) is legal.
>
>   *For example:* When `FOF` = 3:
>   - Delta header value -3 is legal
>   - Delta header value +5 is illegal

- **DestReg[2:0]:** Previous frame-offset value in flit granularity (32B).
- The operation is asynchronous, the program must SYNC on `lflag` before relying on re-base results (e.g., can't send out or copy before SYNC).

Timing Characteristics: Asynchronous operation.

4.13.44 SETREFCNT

Function: Set the reference count of a packet header buffer.

Syntax

```
SETREFCNT BuffIndexImm, RefCntImm
```

Instruction Format

SETREFCNT BuffIndexImm, RefCntImm

Instruction Operation: Set the reference-counter value of a packet header buffer.

- **BuffIndexImm:** Range: Logical buffer indices: 0 or 1.
- **RefCntImm:** Range: 0, 1.

Timing Characteristics: Asynchronous operation

4.13.45 GETFPTR

Function: Get the current packet (frame) header pointers, and the header size.

Syntax

```
GETFPTR Rd.<4Bsel>
```

Instruction Format

GETFPTR DestReg

Instruction Operation:

Get the HW buffer indices 0 and 1 of the packet (frame) header, and the header size, and store in DestReg.

DestReg is a 4B register with the following bits:		
Bits	[2:0]	Buffer 0 HW index
	[3:3]	Buffer 0 valid bit
	[6:4]	Buffer 1 HW index
	[7:7]	Buffer 1 valid bit
	[25:16]	Current header-size. Original header size max = 256 byte. However, with the addition of a tunnel header, the header-size may be larger (up to 256+256 bytes).

Timing Characteristics: Asynchronous operation

4.13.46 REPARSE

Function: Send packet to parser for reparsing.

Syntax

```
REPARSE.LF<N> Rs.<4Bsel>
```

Instruction Format

REPARSE.LF<N> SrcReg

Instruction Operation: Send packet to parser for reparsing.

Reparse is an asynchronous operation, and the MAP program can continue while the packet is being reparsed. During this time, the MAP program must not perform any operation that may interfere with the reparsing program, such as **FREBASE, CPH, STH,** etc.

The parser supports parsing of the header-size as received internally from the MAP, relative to the **FOF** (Frame Offset). The header size is limited to 256 bytes.

In reparse scenario, the parser does not preload the `HDR.PRSENT`, and `HDR.OFFSET` registers; `R1`, and `R7`, as well as the Standard Metadata (SMD), are left intact and not modified by the parser. The developer must ensure, by design, that the MAP program designates one or more registers in which the parser program can store the reparsed results.

During execution, the reparsing program results are stored in internal parser storage. The results include `HDR.PRESENT` and `HDR.OFFSET` fields, and SMD status fields.
When the reparsing is completed, the results are written by the parser program to the pre-designated MAP registers.

When the reparsing is completed, the LFLAG<N> is set by the parser. The MAP program must perform a **SYNC** on the LFLAG in order to continue. In this scenario, `R7.0` is not preloaded, however a copy of the SMD first 4 bytes are available to the MAP as part of the parser results (copied to MAP registers by the parser program).
Note that the port-type field in this copy is not applicable.

In case of abnormal program termination due to parsing failures, status flags are set in the SMD. The parser program, however, always returns to the MAP SYNC point, even when the parser program ends with a DROP instruction. For such cases, the SYNC results with a failure.

- **SrcReg[7:0]:** Cursor delta value. An unsigned value in bytes, which specifies the positive delta from the frame offset, from which to start reparsing.

  Range: 0 to 128. The parser sets the initial cursor value to this offset, and `HDR.OFFSET` values in the reparsing program are set relative to the frame offset.

- **SrcReg[15:8]:** A cookie value.
  This value is conveyed to the parser program in parser register `R3[127:120]`.

Timing Characteristics: Asynchronous operation

4.13.47 IREQ

Function: Generate interrupt request

Syntax

```
IREQ IntNumberImm
```

Instruction Format

IREQ IntNumberImm

Instruction Operation: Generate interrupt request toward the host CPU.

- **IntNumberImm:** 0 to 7

Timing Characteristics: Single cycle

4.13.48 IRETCURR, IRETNEXT

Function: Return from exception/soft interrupt.

Syntax

IRETCURR: `IRETCURR`

IRETNEXT: `IRETNEXT`

Instruction Format

IRETCURR: IRETCURR

IRETNEXT: IRETNEXT

Instruction Operation: Return from exception handler to main MAP program.

- **IRETCURR:** The program returns to the same address from which the jump was taken.
- **IRETNEXT:** The program returns to the next address.

These return instructions can be used only when the MAP program is interrupted and executes exception handling code in debug mode.

Timing Characteristics: Single cycle

4.13.49 SWI

Function: Invoke a soft interrupt

Syntax

```
SWI
```

Instruction Format

SWI

Instruction Operation: Execute a software exception.

The program saves the context and jumps to the exception entry point.

Timing Characteristics: Single cycle

4.13.50 WAIT

Function: Stall the MAP program

Syntax

```
WAIT
```

Instruction Format

WAIT

Instruction Operation: The program stops execution and stalls.

The control-plane can resume the program.

- The WAIT instruction can only be used within a software exception code, and it must be followed by the **IRETNEXT/CURR** instruction in order to return to the normal program flow.
- While the program stalls, all asynchronous operations continue normally.

Timing Characteristics: Single cycle

4.13.51 NOP

Function: No operation

Syntax

```
NOP
```

Instruction Format

Instruction Operation:

Timing Characteristics: Single Cycle

4.13.52 SIZEQUERY

Function: Query Packet Size

Syntax

```
SIZEQUERY.LF<N> Rd.<4Bsel>
```

Instruction Format

SIZEQUERY.query-result-flag PacketSizeReg

Instruction Operation: Query the size of the handled packet.

The operation is asynchronous, and must be finalized by the MAP program by a **SYNC** on the LFLAG.

- The MAP program must handle operation success or failure (by providing a branch-to-PC to the **SYNC** or **SYNC.N** upon success/failure.)
- **PacketSizeReg[13:0]**: Packet byte size.

Timing Characteristics: Asynchronous operation.

4.13.53 MCREQUEST

Function: Request multicast contexts in multicast flow.

Syntax

```
MCREQUEST.LF<N>
```

Instruction Format

MCREQUEST.result-flag

Instruction Operation:

The operation is relevant to multicast flow

- The operation is asynchronous, and is finalized by the MAP program by executing **SYNC.N** on the LFLAG.
- If the result is negative, the MAP program may choose to wait and request again, or abort the multicast flow.

Timing Characteristics: Asynchronous operation.

4.13.54 MCDONE

Function: Finish a multicast operation.

Syntax

```
MCDONE.Rs<4Bsel>
```

Instruction Format

MCDONE NumCopiesReg

Instruction Operation: Inform HW that the multicast process for the current packet is done, and report the total number of copies sent. From this point on, the MAP program must not read nor modify the header data.

NumCopiesReg[12:0]: A 4B register containing the total number of copies sent.

- Number of copies is limited to 8K-1.
- The operation is asynchronous, however **SYNC** is not required.
- The operation must be executed in multicast flow, even if no copies were generated.

Timing Characteristics: Asynchronous operation.

Contact Information