6 SHIFTER

The ADSP-TS201 TigerSHARC processor core contains two computation units known as compute blocks. Each compute block contains a register file and four independent computation units—an ALU, a CLU, a multiplier, and a shifter. The shifter is highlighted in Figure 6-1. The shifter takes its inputs from the register file, and returns its outputs to the register file.

This chapter provides:

The shifter performs *bit wise operations* (arithmetic and logical shifts) and performs *bit field operations* (field extraction and deposition) for the processor. The shifter also executes *data conversion operations* such as fixed- and floating-point format conversions. Shifter operations include:

- Shift and rotate bit field, from off-scale left to off-scale right
- Bit manipulation; bit set, clear, toggle, and test
- Bit field manipulation; field extract and deposit
- Scaling factor identification, 16-bit block floating-point
- Extract exponent
- Count number of leading ones or zeros

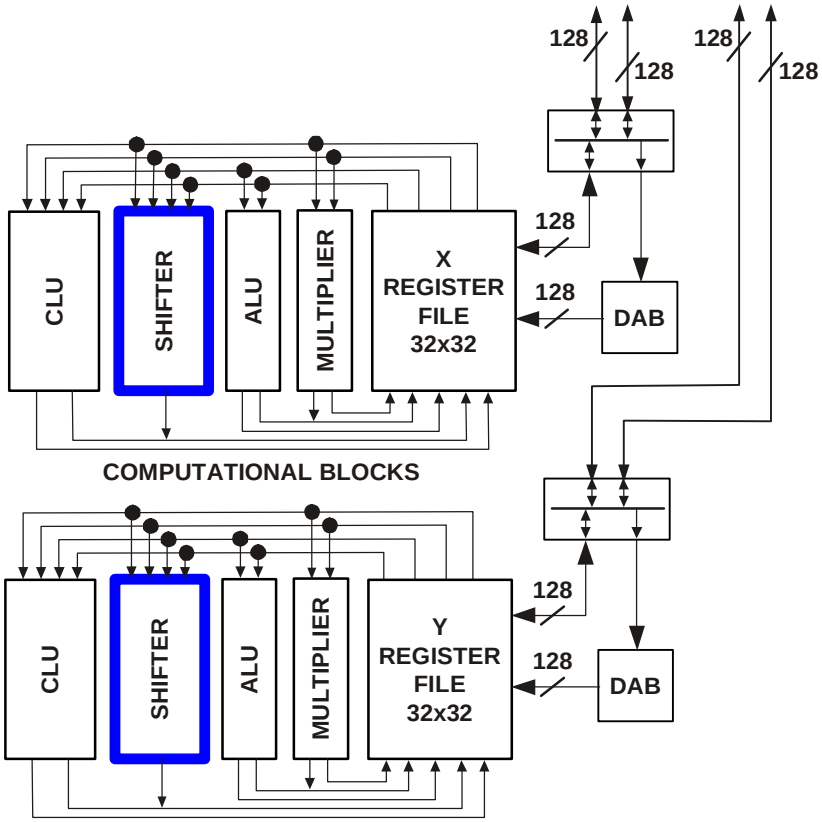


Figure 6-1. Shifters in Compute Block X and Y

The shifter operates on fixed-point data and can take the following as input:

- One long word (64-bit) operand
- One or two normal word (32-bit) operands
- Two or four short word (16-bit) operands
- Four or eight byte word (8-bit) operands

As shown in Figure 6-1, the shifter has four inputs and four outputs (unlike the ALU and multiplier, which have two inputs and outputs). The shifter's I/O paths within the compute block have some implications for instruction parallelism.

- Shifter instructions that use three inputs cannot be executed in parallel with any other compute block operations.
- For `FDEP`, `MASK`, `GETBITS` and `PUTBITS` instructions, there are three registers that are passed into the shifter. This operation uses three compute block ports. The output is being placed in the same port.

For more information on available ports and instruction parallelism, see "Instruction Parallelism Rules" on page 1-26.

Within instructions, the register name syntax identifies the input operand and output result data size and type. For more information on data size and type selection for shifter instructions, see "Register File Registers" on page 2-5.

The remainder of this chapter presents descriptions of shifter instructions and results using instruction syntax. For an explanation of the instruction syntax conventions used in shifter and other instructions, see "Instruction Line Syntax and Structure" on page 1-22. For a list of shifter instructions and their syntax, see "Shifter Instruction Summary" on page 6-19.

Shifter Operations

The shifter operates on one 64-bit, one or two 32-bit, two or four 16-bit, and four or eight 8-bit fixed-point operands. Shifter operations include:

- Shifts and rotates from off-scale left to off-scale right
- Bit manipulation operations, including bit set, clear, toggle and test

- Bit field manipulation operations, including field extract and deposit, using register BFOTMP (which is internal to the shifter)
- Bit FIFO operations to support bit streams with fields of varying length
- Support for ADSP-2100 family compatible fixed-point and floating-point conversion operations (such as exponent extract, number of leading 1s or 0s)

The shifter operates on the compute block register files and operates on the shifter register BFOTMP—an internal shifter register which is used for the PUTBITS instruction. Shifter operations can take their *Rm* input (data operated on) from the register file and take their *Rn* input (shift magnitudes) either from the register file or from immediate data provided in the instruction. In cases where the operation involves a third input operand, *Rm* and *Rn* inputs are taken from the register file, and the third input, *Rs*, is a read-modify-write (RMW).

Shift magnitudes for register file-based operations—where the shift magnitude comes from *Rn*—are held in the right-most bits of *Rn*. The shift magnitude size (number of bits) varies with the size of the output operand, where *Rn* is 8 bits for long word output, 7 bits for normal word output, 6 bits for short word output and 6 bits for byte word output. In this way, full-scale right and left shifts can be achieved. Bits of *Rn* outside of the shift magnitude field are masked.

The following sections describe shifter operation details:

- "Logical Shift Operation" on page 6-5
- "Arithmetic Shift Operation" on page 6-6
- "Bit Manipulation Operations" on page 6-7
- "Bit Field Manipulation Operations" on page 6-8

Logical Shift Operation

The following instruction is an example of a logical shift (LSHIFT). The operation shifts the contents of the XR4 register by the shift value (number of bits) specified in the XR3 register. The shifter places the result in the XR5 register. Figure 6-2 shows how the bits in register XR5 are placed for shift values of 4 and –4.

```
XR5 = LSHIFT R4 BY R3;;
```

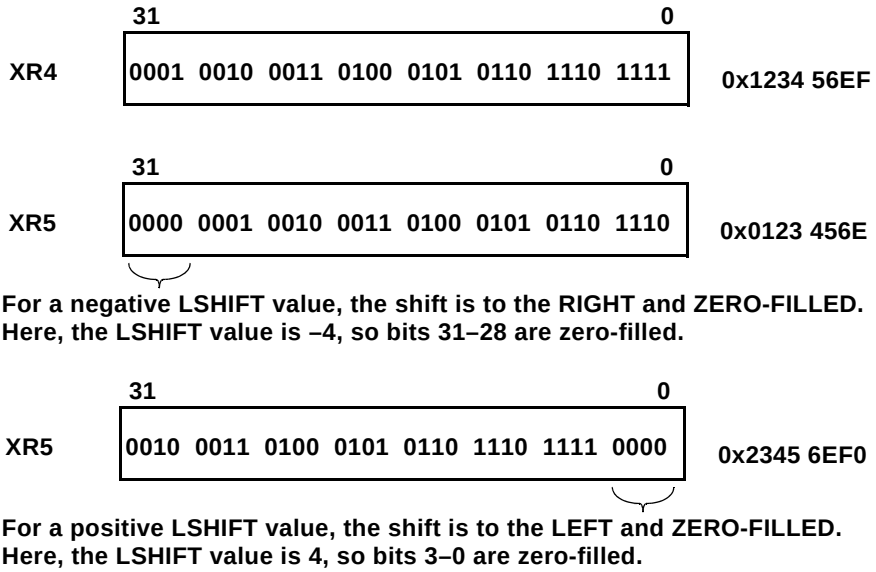


Figure 6-2. LSHIFT Instruction Example

Arithmetic Shift Operation

The following instruction is an example of an arithmetic shift (ASHIFT). The operation shifts the contents of the XR4 register by the shift value (number of bits) specified in the XR3 register. The shifter places the result in the XR5 register. Figure 6-3 shows how the bits in register XR5 are placed for shift values of 8 and –8.

```
XR5 = ASHIFT R4 BY R3 ;;
```

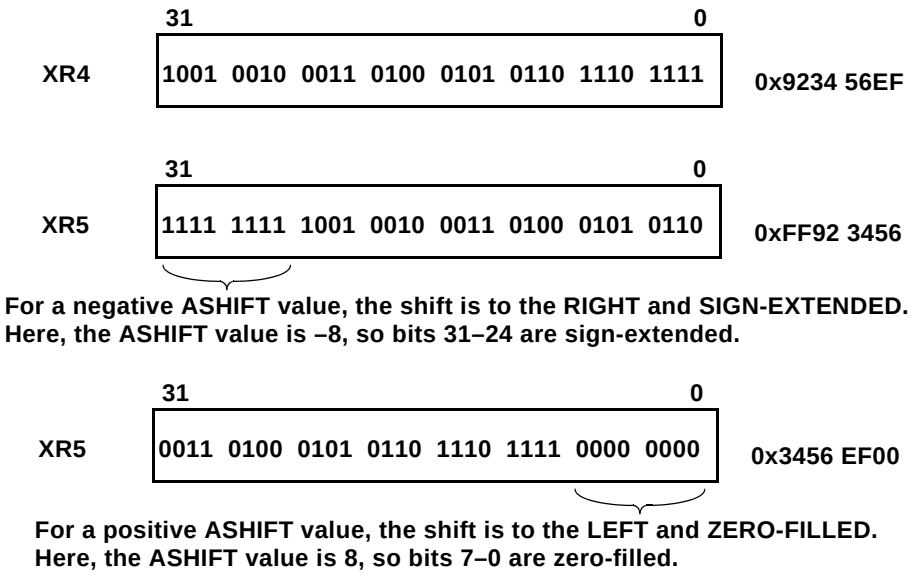


Figure 6-3. ASHIFT Instruction Example

Bit Manipulation Operations

The shifter supports bit manipulation operations including bit clear (BCLR), bit set (BSET), bit toggle (BTGL), and bit test (BITEST). The operand size can be a normal word or a long word. For example:

```
R5 = BCLR  R3  By R2 ;; /* 32-bit operand */
R5:4 = BSET  R3:2  By R6 ;; /* 64-bit operand */
```

The following instruction is an example of bit manipulation (BCLR). The shifter clears the bit in the XR4 register indicated by the bit number specified in the XR3 register. The shifter places the result in the XR5 register. Figure 6-4 shows how the bits in register XR5 are affected for bit number 8.

```
XR5 = BCLR  R4  By R3 ;;
```

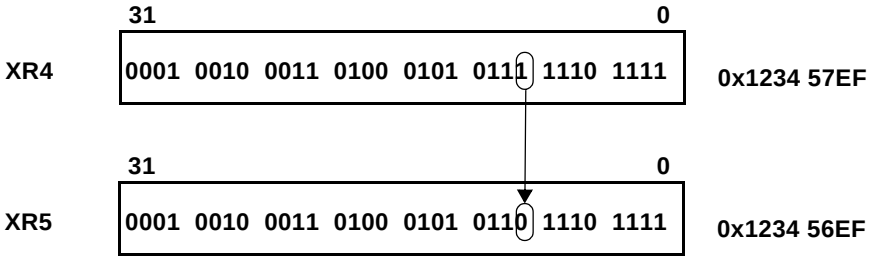


Figure 6-4. BCLR Instruction Example

Bit Field Manipulation Operations

The shifter supports bit field manipulation operations including:

- FEXT–Extracts a field from a register according to the length and position specified by another register
- FDEP–Deposits a right-justified field into a register according to the length and position specified by another register
- MASK–Copies a 32- or 64-bit field created by a mask
- XSTAT/YSTAT–Loads or stores all bits or 14 LSBs only of the XSTAT or YSTAT register

For field extract and deposit operations, the *Rn* operand contains the control information in two fields: <Len7> and <Pos8>. These fields select the number of bits to extract (Len7) and the starting position in *Rm* (Pos8). The location of these fields depends on whether *Rn* is a single- or dual-register as shown in Figure 6-5.

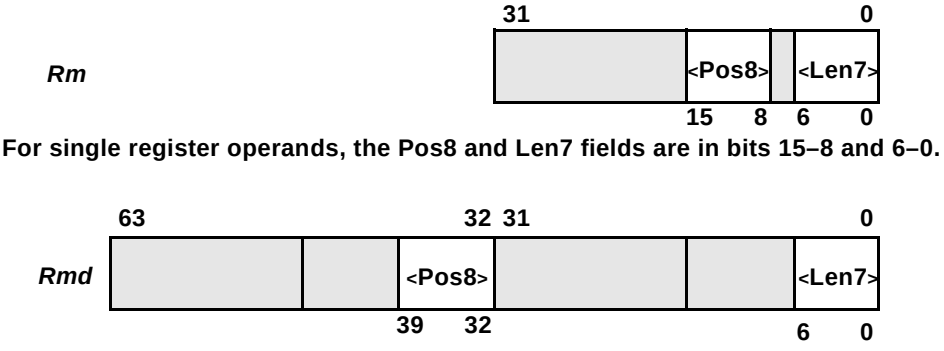
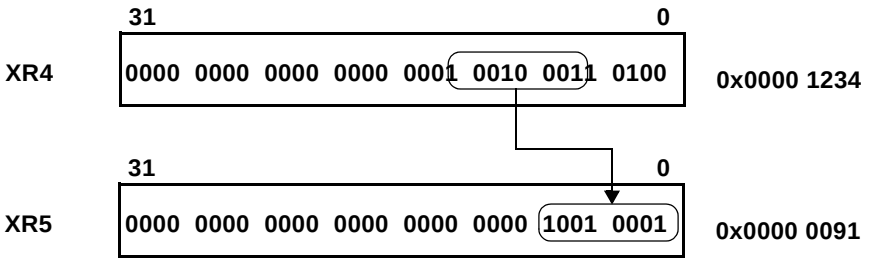


Figure 6-5. FEXT and FDEP Instructions Pos8 and Len7 Fields

There are two versions of the FEXT and FDEP is instructions. One version takes the control information from a register pair. The other version takes control information from a single register. The FEXT instruction takes the data from the indicated position in the source register and places right-justified data in the destination register (*Rs*). The FDEP instruction takes the right-justified data from the source register and places data in the indicated position in the destination register (*Rs*).

The following instruction is an example of bit field extraction (FEXT). The shifter extracts the bit field in the XR4 register indicated by the field position (Pos8) and field length (Len7) values specified in the XR3 register. The shifter places the right-justified result in the XR5 register. The default operation zero-fills the unused bits in the destination register (XR5 in the example). If the FEXT instruction included the sign extend (SE) option, the most significant bit of the extracted field is extended. Figure 6-6 shows how the bits in register XR5 are affected for field position Pos8=5 and field length Len7=8.

```
XR5 = FEXT  R4  By R3 ;; /* Pos8=5, Len7=8, XR3=0x0000 0508 */
```

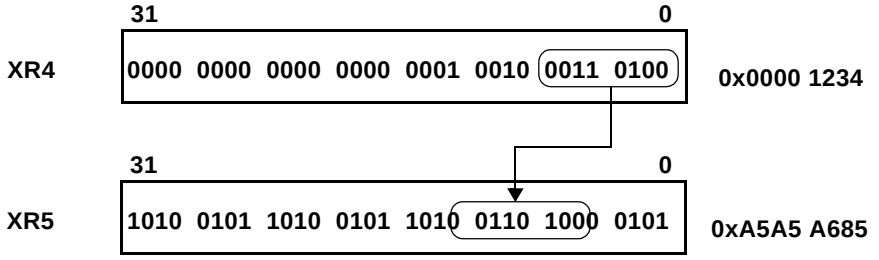


For a FEXT field extraction, the unused bits in the destination are CLEARED unless the SE option is used. Here, bits 31–8 are cleared.

Figure 6-6. FEXT Instruction Example

The following instruction is an example of bit field deposit (FDEP). The shifter extracts the right-justified bit field in the XR4 register field length (Len7) value specified in the XR3 register. The shifter places the result in the XR5 register in the location indicated by the field position (Pos8). The default operation does not alter the unused bits in the destination register (XR5 in the example). If the FDEP instruction included the sign extend (SE) option, the most significant bit of the extracted field is extended. If the FDEP instruction included the sign extend (ZF) option, the most significant unused bits of result register are zero filled. Figure 6-7 shows how the bits in register XR5 are affected for field position Pos8=5 and field length Len7=8.

```
XR5 = FDEP  R4  By R3 ;; /* Pos8=5, Len7=8, XR3=0x0000 0508, XR5
value before instruction was 0xA5A5 A5A5 */
```



For a FDEP field deposit, the unused bits in the destination are UNCHANGED unless the SE or ZF option is used. Here, bits 31–13 and 4–0 are unchanged.

Figure 6-7. FDEP Instruction Example

The following instruction is an example of mask (MASK) operation. The shifter takes the bits from XR4 corresponding to the mask XR3 and ORs them into the XR5 register. The bits of XR5 outside the mask remain untouched.

```
XR3 = 0x00007B00;;
XR4 = 0x50325032;;
```

```
XR5 = 0x85FFFFFF;; /* before mask instruction */
XR5 += MASK R4 BY R3
/* After mask instruction, XR5 = 0x85FF50FF */
```

Bit Field Conversion Operations

The shifter supports fixed- to floating-point conversion operations including:

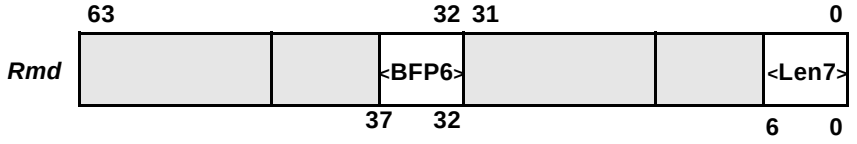
- BKFPT–Determines scaling factor used in 16-bit block floating-point
- EXP–Extracts the exponent
- LDx–Extracts leading zeros (0) or ones (1)

Bit Stream Manipulation Operations

The bit stream manipulation operations, in conjunction with the ALU BFOINC instruction, implement a bit FIFO used for modifying the bits in a contiguous bit stream. The shifter supports bit stream manipulation operations including:

- GETBITS–Extracts bits from a bit stream
- PUTBITS–Deposits bits in a bit stream
- BFOTMP–Temporarily stores or returns overflow from GETBITS and PUTBITS instructions

For bit stream extract (GETBITS) and deposit (PUTBITS) operations, the *Rnd* operand contains the control information in two fields: <BFP6> and <Len7>. These fields in the dual register, *Rnd*, appear in Figure 6-8.



The dual register operand provides the BFP6 and Len6 fields (bits 37–32 and 5–0). Note that the BFP must be incremented using the ALU's BFOINC instruction.

Figure 6-8. GETBITS and PUTBITS Instructions BFP6 and Len7 Fields

The GETBITS instruction extracts the number of bits indicated by Len7 starting at BFP6 and places the right-justified data in the output register. The unused bits are cleared unless the sign extend (SE) option is used. With the SE option, the most significant bit of the extract is extended to the most significant bit of the output register.

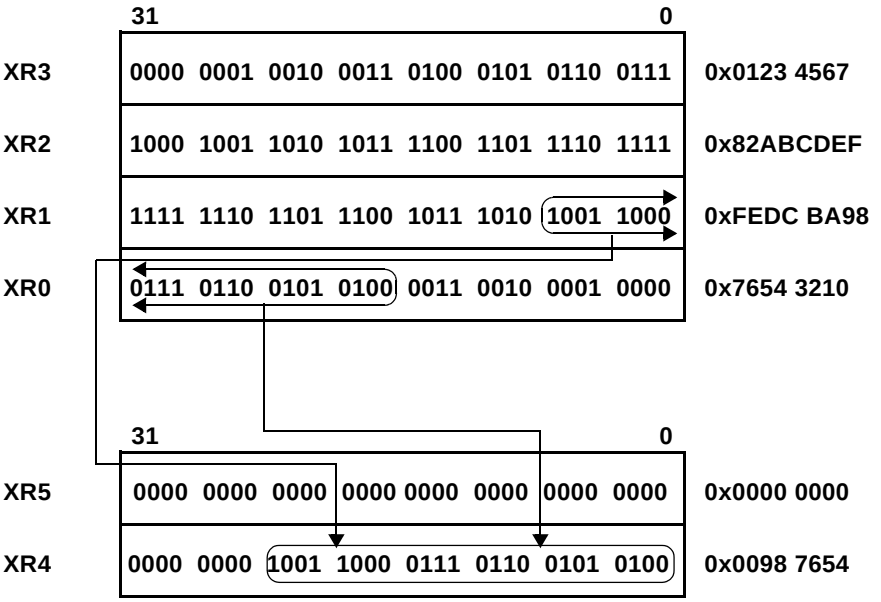
The following instruction is an example of bit stream extraction (GETBITS). The shifter extracts a portion of the bit stream in the XR3:0 quad register indicated by the bit FIFO position (BFP6) and field length (Len7) values specified in the XR7:6 dual register.

Use the ALU's BFOINC instruction to increment the bit FIFO pointer. Normally, an update of bit FIFO pointer is necessary after executing GETBITS. The ALU instruction BFOINC adds BFP6 and Len7 fields, divides them by 64 and returns the remainder to BFP6 field. If for example, BFP6 is 0x30 and Len7 is 0x18, the new value of BFP6 is 0x08 and the flag AN in XSTAT register is set. This flag may be used to identify this situation and proceed accordingly.

In the example, the shifter places the right-justified result in the XR5:4 dual register. The default operation zero-fills the unused bits in the destination register (XR5:4 in the example). If the GETBITS instruction included

the sign extend (SE) option, the most significant bit of the extracted field is extended. Figure 6-9 shows how the bits in register XR5:4 are affected for field position BFP6=16 and field length Len7=24.

```
XR5:4 = GETBITS R3:0 BY R7:6 ;;
/* BFP6=16, Len7=24, XR7:6=0x0000 0010 0000 0018 */
```



For a GETBITS field extraction, the unused bits in the destination are CLEARED unless the SE option is used. Here, bits XR5 and bits 31–24 of XR4 are cleared.

Figure 6-9. GETBITS Instruction Example

The PUTBITS instruction deposits the 64 bits from *Rmd* registers into a contiguous bit stream held in the quad register composed of BFOTMP in the top and *Rsd* in the bottom. In PUTBITS, the BFP field specifies the starting bit where the insertion begins in *Rsd* register, but the Len7 field is ignored. Update of BFP may only be performed by the ALU with the instruction BFOINC.

The following instruction is an example of bit stream placement (PUTBITS). The shifter puts the content of the registers XR3:2 into the bit FIFO composed by XR5:4 and BFOTMP beginning with bit 16 of XR4 (specified into BFP field of XR7).

```
XR3 = 0x01234567 ;;
XR2 = 0x89abcdef ;;
XR5 = 0x0 ;
XR4 = 0x0 ;
XR5:4 += PUTBITS R3:2 BY R7:6 ;; /* BFP6=16, XR7:6=0x0000 0010
0000 0018 */
/* After PUTBITS instruction, the registers hold:
   xBFOTMP = 0x0000 0000 0000 0123
   XR5     = 0x4567 89ab
   XR4     = 0xcdef 0000
```

Shifter Instruction Options

Some of the shifter instructions have options associated with them that permit flexibility in how the instructions execute. It is important to note that these options modify the detailed execution of instructions, and the options that are particular to a group of instructions—not all options are applicable to all instructions. Instruction options appear in parenthesis at the end of the instruction's slot.

For a list indicating which options apply for particular shifter instructions, see "Shifter Instruction Summary" on page 6-19. The shifter instruction options include:

- () zero filled, right justified
- (SE) sign extended; applies to FEXT, FDEP, and GETBITS instructions
- (ZF) zero filled; applies to FDEP instruction

The following are shifter instructions that demonstrate bit field manipulation operations with options applied.

```
XR5 = FEXT  R4  By R3 (SE) ;;
 /* The SE option in this instruction sets bits 31–8 to 1 in Figure 6-6 on page 6-9 */

XR5 = FDEP  R4  By R3 (ZF) ;;
 /* The ZF option in this instruction clears bits 31–13 and 4–0 in Figure 6-7 on page 6-10 */
```

Sign Extended Option

The sign extend `(SE)` option is available for the `FEXT`, `FDEP`, and `GETBITS` shifter instructions. If used, this option extends the value of the most significant bit of the placed bit field through the most significant bit of the output register.

Zero Filled Option

The zero filled `(ZF)` option is available only for the `FDEP` instruction. If used, this option clears all the unused bits above the most significant bit of the placed bit field in the output register.

Shifter Execution Status

Shifter operations update status flags in the compute block's Arithmetic Status (`XSTAT` and `YSTAT`) register (see Figure 2-2 on page 2-4 and Figure 2-3 on page 2-5). Programs can use status flags to control execution of conditional instructions and initiate software exception interrupts. For more information, see "Shifter Execution Conditions" on page 6-16.

Table 6-1 shows the flags in XSTAT or YSTAT that indicate shifter status (a 1 indicates the condition) for the most recent shifter operation.

Table 6-1. Shifter Status Flags

Flag	Definition	Updated By…
SZ	Shifter fixed-point zero	All shifter ops
SN	Shifter negative	All shifter ops
BF1–0	Shifter block floating-point	BKFPT instruction only

Flag update occurs at the end of each operation and is available on the next instruction slot. A program cannot write the arithmetic status register explicitly in the same cycle that the ALU, CLU or multiplier are performing an operation.

Multi-operand instructions (for example, B*Rs* = ASHIFT *Rn* BY *Rm*;) produce multiple sets of results. In this case, the processor determines a flag by ORing the result flag values from individual results.

Shifter Execution Conditions

In a conditional shifter instruction, the execution of the entire instruction line can depend on the specified condition at the beginning of the instruction line. Conditional shifter instructions take the form:

```
IF cond; DO, instr.; DO, instr.; DO, instruct. ;;
```

This syntax permits up to three instructions to be controlled by a condition. Omitting the DO before the instruction makes the instruction unconditional.

Table 6-2 lists the shifter conditions. For more information on conditional instructions, see "Conditional Execution" on page 8-12.

Table 6-2. Shifter Conditions

Condition	Description	Flags set
SEQ	Equal to zero	SZ = 1
SLT	Less than zero	SN = 1 and SZ = 0
NSEQ	Not equal to zero	SZ = 0
NSLT	Not less than zero	SN = 0 and SZ = 1

Shifter Static Flags

In the program sequencer, the static flag (`SFREG`) can store status flag values for later usage in conditional instructions. With `SFREG`, each compute block has two dedicated static flags `X/YSCF0` (condition is `SF0`) and `X/YSCF1` (condition is `SF1`). The following example shows how to load a compute block condition value into a static flag register.

```
XSCF0 = XSEQ ;; /* Load X-compute block SEQ flag into XSCF0 bit
in static flags (SFREG) register */
IF SF0, XR5 = LSHIFT R4 BY R3 ;; /* the SF0 condition tests the
XSCF0 static flag */
```

For more information on static flags, see "Conditional Execution" on page 8-12.

Shifter Examples

Listing 6-1 provides a number of shifter instruction examples. The comments with the instructions identify the key features of the instruction, such as input operand size and register usage.

Listing 6-1. Shifter Instruction Examples

```
XR5 = LSHIFT R4 BY R3;;
/* This is a logical shift of register XR4 by the value contained
in XR3. */

YR1 = ASHIFT R2 BY R0;;
/* This is an arithmetic shift of register XR2 by the value con-
tained in XR0. */

R1:0 = ROT R3:2 BY 8;;
/* This instruction rotates the content of R3:2 in both the X-
and Y-ALUs by 8 and places the result in XR1:0 and YR1:0. */

XBITEST R1:0 BY R7;;
/*  This instruction tests the bit indicated in XR7 of XR1:0 and
sets accordingly the flags XSZ and XSN in XSTAT. */

R9:8 = BTGL R11:10 BY R13;;
/* This instruction toggles the bit indicated in R13 of XR11:10
and YR11:10 and puts the result in xR9:8 and yR9:8. */

XR15 = LDO R17;;
/* This instruction extracts the leading number of zeros of xR17
and places the result into xR15. */
```

Shifter Instruction Summary

Listing 6-2 shows the shifter instructions' syntax. The conventions used in these listings for representing register names, optional items, and choices are covered in detail in "Register File Registers" on page 2-5. Briefly, these conventions are:

- `{ }` – the curly braces enclose options; these braces are not part of the instruction syntax.
- `|` – the vertical bar separates choices; this bar is not part of the instruction syntax.
- *Rmd* – the register names in italic represent user selectable single (*Rs*, *Rm*, *Rn*), double (*Rsd*, *Rmd*, *Rnd*) or quad (*Rsq*, *Rmq*, *Rnq*) register names.

In shifter instructions, output register name relates to output operand size as follows:

- The L prefix on an output operand (register name) indicates long word (64-bit) output. For example, the following instruction syntax indicates single, long word output:

  ```
  LRsd = ASHIFT Rmd BY Rnd;
  ```

- The absence of a prefix on an output operand (register name) indicates normal word (32-bit) output. For example, the following instruction syntax indicates single, normal word output:

  ```
  Rs = ASHIFT Rm BY Rn;
  ```

- A dual register name on an output operand (register name) indicates two normal word (32-bit) outputs. For example, the following instruction syntax indicates two, normal word outputs:

  ```
  Rsd = ASHIFT Rmd BY Rnd;
  ```

- The S prefix on an output operand (register name) indicates two or four short word (16-bit) outputs. For example, the following instruction syntax indicates two or four, short word outputs:

```
SRs = ASHIFT Rm BY Rn /* two outputs */;
SRsd = ASHIFT Rmd BY Rnd; /* four outputs */
```

- The B prefix on an output operand (register name) indicates four or eight byte word (8-bit) outputs. For example, the following instruction syntax indicates four or eight, byte word outputs:

```
BRs = ASHIFT Rm BY Rn /* four outputs */;
BRsd = ASHIFT Rmd BY Rnd; /* eight outputs */
```

Each instruction presented here occupies one instruction slot in an instruction line. For more information about instruction lines and instruction combination constraints, see "Instruction Line Syntax and Structure" on page 1-22 and "Instruction Parallelism Rules" on page 1-26.

Listing 6-2. Shifter Instructions

```
{X|Y|XY}{B|S}Rs = LSHIFT|ASHIFT Rm BY Rn|<Imm> ;[1,2]
{X|Y|XY}{B|S|L}Rsd = LSHIFT|ASHIFT Rmd BY Rn|<Imm> ;[1,2]

{X|Y|XY}Rs = ROT Rm BY Rn|<Imm6> ;[1]
{X|Y|XY}{L}Rsd = ROT Rmd BY Rnd|<Imm> ;[1,2]

{X|Y|XY}Rs = FEXT Rm BY Rn|Rnd {(SE)} ;[3]
{X|Y|XY}LRsd = FEXT Rmd BY Rn|Rnd {(SE)} ;[3]

{X|Y|XY}Rs += FDEP Rm BY Rn|Rnd {(SE|ZF)} ;[3]
{X|Y|XY}LRsd += FDEP Rmd BY Rn|Rnd {(SE|ZF)} ;[3]

{X|Y|XY}Rs += MASK Rm BY Rn ;
{X|Y|XY}LRsd += MASK Rmd BY Rnd ;

{X|Y|XY}Rsd = GETBITS Rmq BY Rnd {(SE)} ;

{X|Y|XY}Rsd += PUTBITS Rmd BY Rnd ;

{X|Y|XY}BITEST Rm BY Rn|<Imm5> ;
{X|Y|XY}BITEST Rmd BY Rnd|<Imm6> ;

{X|Y|XY}Rs = BCLR|BSET|BTGL Rm BY Rn|<Imm5> ;
{X|Y|XY}Rsd = BCLR|BSET|BTGL Rmd BY Rn|<Imm6> ;

{X|Y|XY}Rs = LD0|LD1 Rm|Rmd ;

{X|Y|XY}Rs = EXP Rm|Rmd ;

{X|Y}STAT = Rm ;
```

---

[1] The *Rn* data size (bits) for the shift magnitude varies with the output operand: Byte: 5, Short: 6, Normal: 7, Long: 8.

[2] The size in bits of the *Imm* data varies with the output operand: Byte: 4, Short: 5, Normal: 6, Long: 7.

[3] The placement of the Pos8 and Len7 fields varies with the *Rn*/*Rnd* register, see Figure 6-5 on page 6-8.

```
{X|Y}STATL = Rm ;
{X|Y}Rs = {X|Y}STAT ;

{X|Y|XY}BKFPT Rmd, Rnd ;

{X|Y|XY}Rsd = BFOTMP ;
{X|Y|XY}BFOTMP = Rmd ;
```