

# 5 SHIFTER INSTRUCTIONS

The instruction set provides shifter instructions for performing shift operations on 16-bit input to yield 40-bit output. Combining these functions, programs can efficiently implement numeric format control, including full floating-point representation. Shifter operations include:

- “Arithmetic Shift” on page 5-6
- “Arithmetic Shift Immediate” on page 5-8
- “Logical Shift” on page 5-10
- “Logical Shift Immediate” on page 5-12
- “Normalize” on page 5-14
- “Normalize Immediate” on page 5-17
- “Exponent Derive” on page 5-20
- “Exponent (Block) Adjust” on page 5-23

This chapter describes the individual shifter instructions and the following related topics:

- “Shifter Registers” on page 5-2
- “Shifter Instruction Options” on page 5-3
- “Shifter Status Flags” on page 5-5
- “Denormalization” on page 5-26

## Shifter Instructions

For details on condition codes, see [“Condition Code \(CCODE\) Register” on page 2-6](#).

## Shifter Registers

As shown in [Table 5-1](#), the shifter has five registers.

Table 5-1. Summary of shifter registers

| Name | Size      | Description   |
|------|-----------|---|
| SR0  | 16 bits   | Shifter Result register (low word). SR denotes SR0, SR1, and SR2 combined, which hold the 40-bit shifter result.  |
| SR1  | 16 bits   | Shifter Result register (middle word). This register also functions as the multiplier's y-input feedback register. SR denotes SR0, SR1, and SR2 combined, which hold the 40-bit shifter result.   |
| SR2  | 16/8 bits | Shifter Result register (high byte). SR denotes SR0, SR1, and SR2 combined, which hold the 40-bit shifter result. Although this register is 16 bits wide, for shifter operations, only the lower eight bits are used.   |
| SB   | 16/5 bits | <p>Shifter Block exponent register. Although this register is 16 bits wide, for shifter operations, only the lower five bits are used.</p> <p>Contains the effective exponent derived from the number with greatest magnitude in a block of numbers. This value provides the shift code for all numbers in the block in subsequent NORM or xSHIFT instructions.</p> <p>The value in this register is sign-extended to form a 16-bit value when transferred to memory or to other data registers.</p> <p>Nonshifter instructions can use SB for a 16-bit scratch register.</p> |

Table 5-1. Summary of shifter registers (Cont'd)

| Name | Size      | Description   |
|------|-----------|---|
| SE   | 16/8 bits | <p>Shifter Exponent register. Although this register is 16 bits wide, for shifter operations, only the lower eight bits are used.</p> <p>Contains the effective exponent derived from the input data. This value provides the shift code for a subsequent NORM or xSHIFT instruction.</p> <p>The value in this register is sign-extended to form a 16-bit value when transferred to memory or to other data registers.</p> <p>Nonshifter instructions can use SE for a 16-bit scratch register.</p> |
| SI   | 16 bits   | <p>Shifter Input register. Provides single-precision twos-complement input to shifter instructions.</p>   |

When a shifter operation writes data to `SR1`, it sign extends the value into `SR2`, overwriting the previous contents of `SR2`.

The `SB` and `SE` registers are the result registers for the block exponent adjust (`EXPADJ`) operation and derive exponent (`EXP`) operation, respectively. The shifter input register `SI` supplies single-precision input data to any shifter operation, except `EXPADJ`. To input the result from an ALU or MAC operation directly, you use the appropriate result register—`SR`, `AR`, or `MR`.

## Shifter Instruction Options

Almost all shifter instructions have two to three options: (`HI`), (`LO`), and (`HIX`). Each option enables a different exponent detector mode that operates only while the instruction executes. The shifter interprets and handles the input data according to the selected mode.

For the derive exponent (`EXP`) and block exponent adjust (`EXPADJ`) operations, the shifter calculates the shift code—the direction and number of bits to shift—then stores that value in the `SE` register. For the `ASHIFT`, `LSHIFT`, and `NORM` operations, the user can supply the value of the shift

## Shifter Instructions

code directly to the SE or SB registers or use the result of a previous EXP or EXPADJ operation.

For the ASHIFT, LSHIFT, and NORM operations:

- (HI) Operation references the upper half of the output field.
- (LO) Operation references the lower half of the output field.

For the exponent derive (EXP) operation:

- (HIX) Use this mode for shifts and normalization of results from ALU operations.

Input data is the result of an add or subtract operation that may have overflowed. The shifter examines the ALU overflow bit AV. If AV=1, the effective exponent of the input is +1 (this value indicates that overflow occurred before the EXP operation executed). If AV=0, no overflow occurred and the shifter performs the same operations as the (HI) mode.

- (HI) Input data is a single-precision signed number or the upper half of a double-precision signed number. The number of leading sign bits in the input operand, which equals the number of sign bits minus one, determines the shift code.

(By default, the EXPADJ operation always operates in this mode.)

- (LO) Input data is the lower half of a double-precision signed number. To derive the exponent on a double-precision number, you must perform the EXP operation twice, once on the upper half of the input, and once on the lower half. For details, [“Exponent Derive” on page 5-20](#).

### Shifter Status Flags

Two status flags in the `ASTAT` register, `SS` and `SV`, record the status of shifter operations.

`SS`     Records the sign of the shifter input operand

`SS` = 0 positive (+) input

`SS` = 1 negative (–) input

`SV`     Records overflow or underflow status

`SV` = 0 no overflow or underflow occurred

`SV` = 1 overflow or underflow occurred

## Shifter Instructions

### Arithmetic Shift

$$[IF\ COND] \quad SR = [SR\ OR] \quad ASHIFT\ DREG \left( \begin{array}{|c|} \hline HI \\ \hline LO \\ \hline \end{array} \right) ;$$

#### FUNCTION

Arithmetically shifts the bits of the operand by the amount (number of bits) and direction specified by the shift code (value) in the *SE* register. A positive value produces a left (up) shift, and a negative value produces a right (down) shift. Optionally, the shift can be based on a half of the 32-bit output field being shifted. For more information on output options, see “[Shifter Instruction Options](#)” on page 5-3.

If execution is based on a condition, the shifter performs the operation only if the condition evaluates true, and it performs a *NOP* operation if the condition evaluates false. Omitting the condition forces unconditional execution of the instruction.

With the *SR OR* option selected, the shifter ORs the shifted output with the current contents of the *SR* register and stores that value in *SR*. Otherwise, it overwrites the current contents of the *SR* register with the shifted output.

#### INPUT

The input operand, the value to shift, is supplied in a data register. You can use any of these data registers for the *DREG* inputs:

| Register File  |
|--|
| AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SR0, SR1, SR2, SI |

#### OUTPUT

*SR* Shifter result register contains 40-bit result.

**STATUS FLAGS**

| Affected Flags—set or cleared by the operation  | Unaffected Flags               |
|---|--------------------------------|
| SV  | AZ, AN, AV, AC, AS, AQ, SS, MV |
| For information on these status bits in the ASTAT register, see <a href="#">Table 2-2 on page 2-5</a> . |                                |

**DETAILS**

The shifter sign-extends the 40-bit result to the left, replicating the MSB of the input, and it zero-fills the 40-bit result from the right. Bits shifted out past either boundary ( $SR_{39}$  or  $SR_0$ ) are dropped.

To shift a double-precision number, you shift both halves of the input data separately, using the same shift code value for both halves. You `ASHIFT` the upper half of the input data but `LSHIFT` the lower half. The first cycle, you `ASHIFT` the upper half of the input using the `(HI)` option. The second cycle, you `LSHIFT` the lower half using both the `(LO)` and `SR OR` options. Using these options prevents the shifter from sign-extending the MSB of the low word and from overwriting the output (upper word) from the previous `ASHIFT` operation.

**EXAMPLES**

```
AR = 3; SE = AR;          /* shift code, left shift 3 bits */
SI = 0xB6A3;             /* value of upper word of input data */
SR = ASHIFT SI (HI);     /* arithmetically shift high word */
```

**SEE ALSO**

- [“Type 16: Shift Reg0” on page 9-38](#)
- [“Condition Code \(CCODE\) Register” on page 2-6](#)
- [“Mode Status \(MSTAT\) Register” on page 2-11](#)

## Shifter Instructions

### Arithmetic Shift Immediate

$$SR = [SR \text{ OR}] \text{ ASHIFT DREG BY } \langle \text{imm8} \rangle \left( \begin{array}{c} \text{HI} \\ \text{LO} \end{array} \right) ;$$

#### FUNCTION

Arithmetically shifts the bits of the operand by the amount (number of bits) and direction specified by the immediate value. Valid immediate values range from  $-128$  to  $127$ . A positive value produces a left (up) shift, and a negative value produces a right (down) shift. Optionally, the shift can be based on a half of the 32-bit output field being shifted. For more information on output options, see “[Shifter Instruction Options](#)” on page 5-3.

With the `SR OR` option selected, the shifter ORs the shifted output with the current contents of the `SR` register and stores that value in `SR`. Otherwise, it overwrites the current contents of the `SR` register with the shifted output.

#### INPUT

The input operand, the value to shift, is supplied in a data register. You can use any of these data registers for the `DREG` inputs:

| Register File  |
|--|
| AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SR0, SR1, SR2, SI |

#### OUTPUT

`SR` Shifter result register contains 40-bit result.

## STATUS FLAGS

| Affected Flags—set or cleared by the operation  | Unaffected Flags               |
|---|--------------------------------|
| SV  | AZ, AN, AV, AC, AS, AQ, SS, MV |
| For information on these status bits in the ASTAT register, see <a href="#">Table 2-2 on page 2-5</a> . |                                |

## DETAILS

The shifter sign-extends the 40-bit result to the left, replicating the MSB of the input, and it zero-fills the 40-bit result from the right. Bits shifted out past either boundary ( $SR_{39}$  or  $SR_0$ ) are dropped.

To shift a double-precision number, you shift both halves of the input data separately, using the same immediate value for both halves. You `ASHIFT` the upper half of the input data but `LSHIFT` the lower half. The first cycle, you `ASHIFT` the upper half of the input using the `(HI)` option. The second cycle, you `LSHIFT` the lower half using both the `(LO)` and `SR OR` options. Using these options prevents the shifter from sign-extending the MSB of the low word and from overwriting the output (upper word) from the previous `ASHIFT` operation.

## EXAMPLES

```
SI = 0xB6A3;                /* value of upper word of input data */
SR = ASHIFT SI BY 3 (HI); /* arithmetically shift upper word */
```

## SEE ALSO

- [“Type 15: Shift Data8” on page 9-37](#)
- [“Condition Code \(CCODE\) Register” on page 2-6](#)
- [“Mode Status \(MSTAT\) Register” on page 2-11](#)

## Shifter Instructions

### Logical Shift

$$[IF\ COND]\ SR = [SR\ OR]\ LSHIFT\ DREG\ \left( \begin{array}{c} HI \\ LO \end{array} \right) ;$$

#### FUNCTION

Logically shifts the bits of the operand by the amount (number of bits) and direction specified by the shift code (value) in the *SE* register. A positive value produces a left (up) shift, and a negative value produces a right (down) shift. Optionally, the shift can be based on a half of the 32-bit output field being shifted. For more information on output options, see “[Shifter Instruction Options](#)” on page 5-3.

If execution is based on a condition, the shifter performs the operation only if the condition evaluates true, and it performs a *NOP* operation if the condition evaluates false. Omitting the condition forces unconditional execution of the instruction.

With the *SR OR* option selected, the shifter ORs the shifted output with the current contents of the *SR* register and stores that value in *SR*. Otherwise, it overwrites the current contents of the *SR* register with the shifted output.

#### INPUT

The input operand, the value to shift, is supplied in a data register. You can use any of these data registers for the *DREG* inputs:

|  |
|--|
| Register File  |
| AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SR0, SR1, SR2, SI |

#### OUTPUT

*SR* Shifter result register contains 40-bit result.

## STATUS FLAGS

| Affected Flags—set or cleared by the operation  | Unaffected Flags               |
|---|--------------------------------|
| SV  | AZ, AN, AV, AC, AS, AQ, SS, MV |
| For information on these status bits in the ASTAT register, see <a href="#">Table 2-2 on page 2-5</a> . |                                |

## DETAILS

For left shifts (positive shift code), the shifter zero-fills the 40-bit result from the right. Bits shifted out past the high-order boundary ( $SR_{39}$ ) are dropped.

For right shifts (negative shift code), the shifter zero-fills the 40-bit result from the left. Bits shifted out past the low-order boundary ( $SR_0$ ) are dropped.

To shift a double-precision number, you shift both halves of the input data separately, using the same shift code value for both halves. The first cycle, you LSHIFT the upper half of the input using the (HI) option. The second cycle, you LSHIFT the lower half using both the (LO) and SR OR options. Using these options prevents the shifter from overwriting the result (upper word) from the previous LSHIFT operation.

## EXAMPLES

```
AR = 3; SE = AR;           /* shift code left shift 3 bits */
AX0 = 0x765D;             /* value of lower word of input data */
SR = SR OR LSHIFT AX0(LO); /* logically shift low word */
```

## SEE ALSO

- [“Type 16: Shift Reg0” on page 9-38](#)
- [“Condition Code \(CCODE\) Register” on page 2-6](#)
- [“Mode Status \(MSTAT\) Register” on page 2-11](#)

## Shifter Instructions

### Logical Shift Immediate

$$SR = [SR \text{ OR} ] \quad \text{LSHIFT BY } \langle \text{imm8} \rangle \quad \left( \begin{array}{c} \text{HI} \\ \text{LO} \end{array} \right) ;$$

#### FUNCTION

Logically shifts the bits of the operand by the amount (number of bits) and direction specified by the immediate value. Valid immediate values range from  $-128$  to  $127$ . A positive value produces a left (up) shift, and a negative value produces a right (down) shift. Optionally, the shift can be based on a half of the 32-bit output field being shifted. For more information on output options, see [“Shifter Instruction Options” on page 5-3](#).

With the `SR OR` option selected, the shifter ORs the shifted output with the current contents of the `SR` register and stores that value in `SR`. Otherwise, it overwrites the current contents of the `SR` register with the shifted output.

#### INPUT

The input operand, the value to shift, is supplied in a data register. You can use any of these data registers for the `DREG` inputs:

| Register File  |
|--|
| AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SR0, SR1, SR2, SI |

#### OUTPUT

`SR` Shifter result register contains 40-bit result.

## STATUS FLAGS

| Affected Flags—set or cleared by the operation  | Unaffected Flags               |
|---|--------------------------------|
| SV  | AZ, AN, AV, AC, AS, AQ, SS, MV |
| For information on these status bits in the ASTAT register, see <a href="#">Table 2-2 on page 2-5</a> . |                                |

## DETAILS

For left shifts (positive shift code), the shifter zero-fills the 40-bit result from the right. Bits shifted out past the high-order boundary ( $SR_{39}$ ) are dropped.

For right shifts (negative shift code), the shifter zero-fills the 40-bit result from the left. Bits shifted out past the low-order boundary ( $SR_0$ ) are dropped.

To shift a double-precision number, you shift both halves of the input data separately, using the same shift code value for both halves. The first cycle, you LSHIFT the upper half of the input using the (HI) option. The second cycle, you LSHIFT the lower half using both the (LO) and SR OR options. Using these options prevents the shifter from overwriting the result (upper word) from the previous LSHIFT operation.

## EXAMPLES

```
SI = 0xFF6A;                /* single-precision input */
SR = SR OR LSHIFT SI BY 3 (LO); /* logically shift low word */
```

## SEE ALSO

- [“Type 15: Shift Data8” on page 9-37](#)
- [“Condition Code \(CCODE\) Register” on page 2-6](#)
- [“Mode Status \(MSTAT\) Register” on page 2-11](#)

## Shifter Instructions

### Normalize

$$[IF COND] \quad SR = [SR OR] \quad NORM \quad DREG \quad \left( \begin{array}{c} HI \\ LO \end{array} \right) ;$$

#### FUNCTION

Normalization, in essence, is a fixed- to floating-point conversion operation that produces an exponent and a mantissa. Optionally, the operation can be based on a half of the 32-bit output field being shifted. For more information on output options, see [“Shifter Instruction Options” on page 5-3](#).

Normalization using this instruction is a two step process that requires:

- The `EXP` instruction to derive the exponent for the shift code.
- The `NORM` instruction to shift the twos-complement input by the calculated shift code, removing its redundant sign bits and aligning its true sign bit to the high-order bit of the output field.

The `EXP` operation calculates the number of redundant sign bits in the input and stores the negative of that value in `SE`. The `NORM` operation negates the value in `SE` again to generate a positive shift code, ensuring that the input is shifted left.

If execution is based on a condition, the shifter performs the operation only if the condition evaluates true, and it performs a `NOP` operation if the condition evaluates false. Omitting the condition forces unconditional execution of the instruction.

With the `SR OR` option selected, the shifter ORs the shifted output with the current contents of the `SR` register and stores that value in `SR`. Otherwise, it overwrites the current contents of the `SR` register with the shifted output.

## INPUT

The input operand, the value to shift, is supplied in a data register. You can use any of these data registers for the DREG inputs:

|  |
|--|
| Register File  |
| AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SR0, SR1, SR2, SI |

## OUTPUT

SR Shifter result register contains 40-bit result.

## STATUS FLAGS

| Affected Flags—set or cleared by the operation  | Unaffected Flags               |
|---|--------------------------------|
| SV  | AZ, AN, AV, AC, AS, AQ, SS, MV |
| For information on these status bits in the ASTAT register, see <a href="#">Table 2-2 on page 2-5</a> . |                                |

## DETAILS

The (HI) and (LO) options determine how unused bits in the 40-bit output are filled. When the (HI) option is selected, the shifter zero-fills the 40-bit result from the right. When the (LO) option is selected, the shifter zero-fills the 40-bit result to the left. Bits shifted out past the high-order boundary (SR<sub>39</sub>) are dropped.

To normalize a double-precision number, you normalize both halves of the input data separately, using the same shift code value for both halves. First, you use the EXP instruction to derive the exponent to use for the shift code. Then, in the first normalization cycle, you NORM the upper half of the input using the (HI) option. The next cycle, you NORM the lower half using both the (LO) and SR OR options. Using these options prevents the

## Shifter Instructions

shifter from overwriting the result (upper word) from the previous NORM operation.

### EXAMPLES

```
/* Normalize double-precision twos complement data: */
AX1 = 0xF6D4;          /* load hi 2s comp data in dreg */
AX0 = 0x04A2;          /* load lo 2s comp data in dreg */
SE = EXP AX1 (HI);     /* derive exponent on hi word */
SE = EXP AX0 (LO);     /* derive exponent on lo word */
SR = NORM AX1 (HI);    /* normalize hi word */
SR = SR OR NORM AX0 (LO); /* normalize lo word */
```

### SEE ALSO

- [“Type 16: Shift Reg0” on page 9-38](#)
- [“Denormalization” on page 5-26](#)
- [“Condition Code \(CCODE\) Register” on page 2-6](#)
- [“Mode Status \(MSTAT\) Register” on page 2-11](#)

## Normalize Immediate

$$SR = [SR \text{ OR } \text{NORM DREG BY } \langle \text{imm8} \rangle \left( \begin{array}{c} \text{HI} \\ \text{LO} \end{array} \right) ] ;$$

### FUNCTION

Normalization, in essence, is a fixed- to floating-point conversion operation that produces an exponent and a mantissa. Using a positive constant for the shift code, it is a one step process that shifts the twos-complement input left by the specified amount, removing its redundant sign bits and aligning its true sign bit to the high-order bit of the output field. Optionally, the operation can be based on a half of the 32-bit output field being shifted. For more information on output options, see [“Shifter Instruction Options” on page 5-3](#).

With the `SR OR` option selected, the shifter ORs the shifted output with the current contents of the `SR` register and stores that value in `SR`. Otherwise, it overwrites the current contents of the `SR` register with the shifted output.

### INPUT

The input operand, the value to shift, is supplied in a data register. You can use any of these data registers for the `DREG` inputs:

| Register File  |
|--|
| AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SR0, SR1, SR2, SI |

### OUTPUT

`SR` Shifter result register contains 40-bit result.

# Shifter Instructions

## STATUS FLAGS

| Affected Flags—set or cleared by the operation  | Unaffected Flags               |
|---|--------------------------------|
| SV  | AZ, AN, AV, AC, AS, AQ, SS, MV |
| For information on these status bits in the ASTAT register, see <a href="#">Table 2-2 on page 2-5</a> . |                                |

## DETAILS

The (HI) and (LO) options determine how unused bits in the 40-bit output are filled. When the (HI) option is selected, the shifter zero-fills the 40-bit result from the right. When the (LO) option is selected, the shifter zero-fills the 40-bit result to the left. Bits shifted out past the high-order boundary (SR<sub>39</sub>) are dropped.

To normalize a double-precision number, you normalize both halves of the input data separately, using the immediate value for both halves. In the first normalization cycle, you NORM the upper half of the input using the (HI) option. The next cycle, you NORM the lower half using both the (LO) and SR OR options. Using these options prevents the shifter from overwriting the result (upper word) from the previous NORM operation.

## EXAMPLES

```
/* Normalize a double-precision, twos-complement data: */
AX1 = 0xF6D4;          /* load hi 2s comp data in dreg */
AX0 = 0x04A2;          /* load lo 2s comp data in dreg */
SR = NORM AX1 BY 2 (HI); /* normalize hi word */
SR = SR OR NORM AX0 BY 2 (LO); /* normalize lo word */
```

## SEE ALSO

- [“Type 15: Shift Data8” on page 9-37](#)
- [“Denormalization” on page 5-26](#)

- “Condition Code (CCODE) Register” on page 2-6
- “Mode Status (MSTAT) Register” on page 2-11

## Shifter Instructions

### Exponent Derive

```
[ IF COND]          SE = EXP DREG ( | HIX | ) ;
                               | HI  |
                               | LO  |
```

#### FUNCTION

Derives the effective exponent of the input operand to generate the shift code value for use in a subsequent normalization operation. The instruction option, (HIX), (HI), or (LO), determines the resulting shift code. For more information on output options, see [“Shifter Instruction Options” on page 5-3](#).

If execution is based on a condition, the shifter performs the operation only if the condition evaluates true, and it performs a NOP operation if the condition evaluates false. Omitting the condition forces unconditional execution of the instruction.

#### INPUT

The input operand, the value to shift, is supplied in a data register. You can use any of these data registers for the DREG inputs:

| Register File  |
|--|
| AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SR0, SR1, SR2, SI |

#### OUTPUT

SE     Shifter exponent register contains the 8-bit shift code.

**STATUS FLAGS**

| Affected Flags—set or cleared by the operation  | Unaffected Flags               |
|---|--------------------------------|
| SS (Affected by operations using the (HI) and (HIX) options only. Set by the MSB of the input data when AV = 0. In (HIX) mode only, set by the inverted MSB of the input data when AV = 1.) | AZ, AN, AV, AC, AS, AQ, MV, SV |
| For information on these status bits in the ASTAT register, see <a href="#">Table 2-2 on page 2-5</a> .   |                                |

**DETAILS**

You use the (LO) option only to derive the exponent for the low word in a double-precision twos-complement number. But before you do, you must derive the exponent on the high word using either the (HI) or the (HIX) option. The result of the EXP operation on the upper half determines the shift code of the lower half. Unless the upper half contains all sign bits, the SE register contains the correct shift code to use for both EXP (HI/HIX) and (LO) operations. If the upper half does contain all sign bits, EXP (LO) totals the number of sign bits in the double-precision word and stores that value in SE.

**EXAMPLES**

```

/* Normalize double-precision twos complement data: */
AX1 = 0xF6D4;          /* load hi 2s comp data in dreg */
AX0 = 0x04A2;          /* load lo 2s comp data in dreg */
SE = EXP AX1 (HI);     /* derive exponent on hi word */
SE = EXP AX0 (LO);     /* derive exponent on lo word */
SR = NORM AX1 (HI);    /* normalize hi word */
SR = SR OR NORM AX0 (LO); /* normalize lo word */

```

## Shifter Instructions

### SEE ALSO

- [“Type 16: Shift Reg0” on page 9-38](#)
- [“Condition Code \(CCODE\) Register” on page 2-6](#)
- [“Mode Status \(MSTAT\) Register” on page 2-11](#)

## Exponent (Block) Adjust

```
[IF COND] SB = EXPADJ DREG ;
```

### FUNCTION

Derives the effective exponent of the number of largest magnitude in a block of numbers. Using this value for the shift code in subsequent `NORM` instructions, you can normalize each number in the block.

If execution is based on a condition, the shifter performs the operation only if the condition evaluates true, and it performs a `NOP` operation if the condition evaluates false. Omitting the condition forces unconditional execution of the instruction.

### INPUT

The input operand, the value to shift, is supplied in a data register. You can use any of these data registers for the `DREG` inputs:

| Register File  |
|--|
| AX0, AX1, AY0, AY1, AR, MX0, MX1, MY0, MY1, MR0, MR1, MR2, SR0, SR1, SR2, SI |

### OUTPUT

`SB` Shifter block exponent register contains the 5-bit exponent value.

You must initialize `SB` to `-16` before issuing the first `EXPADJ` instruction in the series.

# Shifter Instructions

## STATUS FLAGS

| Affected Flags—set or cleared by the operation  | Unaffected Flags                   |
|---|------------------------------------|
|   | AZ, AN, AV, AC, AS, AQ, SS, MV, SV |
| For information on these status bits in the ASTAT register, see <a href="#">Table 2-2 on page 2-5</a> . |                                    |

## DETAILS

This instruction operates in (HI) mode to derive the exponent. It works on double-precision twos-complement input only. Possible values for the result of the EXPADJ operation range from  $-15$  to  $0$ .

To derive the effective exponent for a block of numbers, you

1. Initialize the SB register to  $-16$ .

```
SB = -16;
```

This value falls below the range of possible exponent values.

2. For each number in the block, derive the effective exponent.

```
SB = EXPADJ DREGx;
```

For the first operation, the shifter derives the exponent and stores it in SB.

For each subsequent operation, the shifter derives the exponent and compares the new value with the current value of SB. If the new value is greater than the current value, the shifter stores the new value in SB, overwriting the old value. Otherwise, the shifter discards the new value and the contents of SB remain unchanged.

At the end of the last EXPADJ operation, SB contains the exponent of the number of largest magnitude in the block.

3. Transfer the contents of SB to SE.

```
SE = SB;
```

SE now contains the shift code to use in subsequent NORM operations to normalize each of the numbers in the block. For details, see [“Normalize” on page 5-14](#).

Alternatively, you can save the exponent in a data register for use later in your program.

### EXAMPLES

```
/* Normalize double-precision twos complement data: */
AX1 = 0xF6D4;          /* load hi 2s comp data in dreg */
AX0 = 0x04A2;          /* load lo 2s comp data in dreg */
SB = -16;              /* initialize SB */
SB = EXPADJ AX1;
SB = EXPADJ AX0;
SE = SB;               /* load block adjusted exp */
SR = NORM AX1 (HI);    /* normalize hi word */
SR = SR OR NORM AX0 (LO); /* normalize lo word */
```

### SEE ALSO

- [“Type 16: Shift Reg0” on page 9-38](#)
- [“Condition Code \(CCODE\) Register” on page 2-6](#)
- [“Mode Status \(MSTAT\) Register” on page 2-11](#)

## Shifter Instructions

### Denormalization

#### FUNCTION

Denormalization is a shift function in which a predefined exponent defines the amount and direction of the shift. In essence, denormalization is a floating- to fixed-point conversion operation. It requires a series of shifter operations:

- The `EXP` instruction to derive the exponent used for the shift code, or `SE` explicitly loaded with the exponent value. `SE` must contain the shift value; for denormalization, you cannot use `ASHIFT/LSHIFT` with an immediate value.
- The `ASHIFT` instruction to shift a single-precision number or the high word of a double-precision number.
- When denormalizing a double-precision number, the `LSHIFT` instruction to shift the low word.

Denormalize a double-precision, twos-complement number:

```
MX1 = -3;           /* generate shift code */
SE = MX1;           /* load value in SE register */
AX1 = 0xB6A3;       /* load high word of input */
AX0 = 0x765D;       /* load low word of input */
SR = ASHIFT AX1(HI); /* arith shift high word */
SR = SR OR LSHIFT AX0(LO); /* logically shift low word */
```

You can reverse shift order, but you must always arithmetically shift the high word of a double-precision number:

```
MX1 = -3;           /* generate shift code */
SE = MX1;           /* load value in SE register */
AX1 = 0xB6A3;       /* load high word of input */
AX0 = 0x765D;       /* load low word of input */
SR = LSHIFT AX0(LO); /* logically shift low word */
SR = SR OR ASHIFT AX1(HI); /* arith shift high word */
```