

6 MULTIFUNCTION INSTRUCTIONS

The instruction set provides multifunction instructions—multiple instructions within a single instruction cycle. Multifunction instructions can perform (in a single cycle) computations in parallel with data move operations. Multifunction instructions are combinations of single instructions delimited with commas and ended with a semicolon, as in:

```
AR = AX0 - AY0, AX0 = MR1; /* ALU sub and reg.-to-reg. move */
```

These operations are the basis for all high-performance DSP functions and take advantage of the DSP's inherent parallelism. Multifunction operations include:

- [“Compute with Dual Memory Read” on page 6-3](#)
- [“Dual Memory Read” on page 6-7](#)
- [“Compute with Memory Read” on page 6-10](#)
- [“Compute with Memory Write” on page 6-14](#)
- [“Compute with Register to Register Move” on page 6-18](#)

This chapter describes each of the multifunction instructions and the order of execution of multifunction operations.

Multifunction instructions combine compute operations with data move operations. The multifunction combinations have no status flags specifically associated with them, but the DSP does update the status flags for the computations that appear within multifunction instructions. For details, see [“Arithmetic Status \(ASTAT\) Register” on page 2-5](#).

Order of Execution of Multifunction Operations

The DSP reads registers and memory at the beginning of the processor pipeline and writes them at the end of it. Normal instruction syntax, read from left to right, implies this functional ordering. For example:

- a. $MR = MR + MX0 * MY0(UU)$, $MX0 = DM(IO += M0)$, $MY0 = PM(I4 += M4)$;
- b. $DM(IO += M0) = AR$, $AR = AX0 + AY0$;
- c. $AR = AX0 - AY0$, $AX0 = MR1$;

This means that, for memory reads, the DSP executes the computation first, using the current value of the input data registers, and then transfers new data from memory or from another data register, overwriting the contents of the data registers (a and c). For memory writes, the DSP transfers the current value from the data register to memory first, and then overwrites the data register with the result of the computation (b).

Even if you alter the order of the operations in your code, execution occurs in the correct order; the assembler issues a warning (if enabled), but results are correct at the opcode level. For example:

```
MX0 = DM(IO += M0), MY0 = PM(I4 += M4), MR = MR + MX0 * MY0(UU);
```

The altered order of operations appears to reverse the order in which the DSP executes the operations, but the DSP always executes instructions using read-first/write-last logic.

The DSP's read-first/write-last logic enables you to use the same data register in more than one multifunction operation. The same data register can serve as an input operand into the computation and as the destination or source register for a data move operation. However, except for the compute with memory write instruction, the same register cannot serve as destination for more than one multifunction operation. Doing so generates unpredictable and erroneous results.

Compute with Dual Memory Read

<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 5px;"><ALU></td></tr> <tr><td style="padding: 2px 5px;"><MAC></td></tr> </table>	<ALU>	<MAC>	,	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 5px;">AX0</td></tr> <tr><td style="padding: 2px 5px;">AX1</td></tr> <tr><td style="padding: 2px 5px;">MX0</td></tr> <tr><td style="padding: 2px 5px;">MX1</td></tr> </table>	AX0	AX1	MX0	MX1	=	DM(<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 5px;">I0</td></tr> <tr><td style="padding: 2px 5px;">I1</td></tr> <tr><td style="padding: 2px 5px;">I2</td></tr> <tr><td style="padding: 2px 5px;">I3</td></tr> </table>	I0	I1	I2	I3	+=	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 5px;">M0</td></tr> <tr><td style="padding: 2px 5px;">M1</td></tr> <tr><td style="padding: 2px 5px;">M2</td></tr> <tr><td style="padding: 2px 5px;">M3</td></tr> </table>	M0	M1	M2	M3),	,	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 5px;">AY0</td></tr> <tr><td style="padding: 2px 5px;">AY1</td></tr> <tr><td style="padding: 2px 5px;">MY0</td></tr> <tr><td style="padding: 2px 5px;">MY1</td></tr> </table>	AY0	AY1	MY0	MY1	=	PM(<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 5px;">I4</td></tr> <tr><td style="padding: 2px 5px;">I5</td></tr> <tr><td style="padding: 2px 5px;">I6</td></tr> <tr><td style="padding: 2px 5px;">I7</td></tr> </table>	I4	I5	I6	I7	+=	<table style="border-collapse: collapse; width: 100%;"> <tr><td style="padding: 2px 5px;">M4</td></tr> <tr><td style="padding: 2px 5px;">M5</td></tr> <tr><td style="padding: 2px 5px;">M6</td></tr> <tr><td style="padding: 2px 5px;">M7</td></tr> </table>	M4	M5	M6	M7);
<ALU>																																										
<MAC>																																										
AX0																																										
AX1																																										
MX0																																										
MX1																																										
I0																																										
I1																																										
I2																																										
I3																																										
M0																																										
M1																																										
M2																																										
M3																																										
AY0																																										
AY1																																										
MY0																																										
MY1																																										
I4																																										
I5																																										
I6																																										
I7																																										
M4																																										
M5																																										
M6																																										
M7																																										

FUNCTION

Combines an ALU or MAC operation with a read from memory over the 16-bit DM data bus and another read from memory over the 24-bit PM data bus. The restricted register forms—*using XOP and YOP registers, not the DREG register file*—of all ALU or MAC instructions are supported, except for the MAC saturate instruction and the divide primitives `DIVS` and `DIVQ`. Also, the multifunction ALU and MAC instructions may not use conditional (IF) options.

The compute operation executes first, using the current contents of the data registers as input operands. Then the memory read operations execute, overwriting the contents of the destination data registers with new data from memory.

The destination of both memory read operations is an ALU or MAC data register. The DM bus read loads an ALU or MAC XOP register, and the PM bus read loads an ALU or MAC YOP register. The memory data is always right-justified in the destination data register.

INPUT


The input operands for the compute operation are specific to the particular operation. For details, see the compute instruction’s individual description.

- For MAC operations, see [“MAC Instructions” on page 4-1](#).
- For ALU operations, see [“ALU Instructions” on page 3-1](#).

Multifunction Instructions

Both data move operations use two DAG registers, index (I_{reg}) and modify (M_{reg}), to generate memory addresses—DAG1 registers for the DM bus access, and DAG2 registers for the PM bus access. For details on DAG registers and data addressing, see [“Data Move Instructions” on page 7-1](#).

- DM/DAG1 $I0, I1, I2, \text{ or } I3$ (index registers)
 $M0, M1, M2, \text{ or } M3$ (modify registers)
- PM/DAG2 $I4, I5, I6, \text{ or } I7$ (index registers)
 $M4, M5, M6, \text{ or } M7$ (modify registers)


 You can use any index register with any modify register from the same DAG. You cannot pair a DAG1 register with a DAG2 register.

OUTPUT

The result register for the compute operation is always the computation unit’s result registers.

AR ALU operations

MR MAC operations

 SR is not a MAC result register for this multifunction instruction.

The destination register for both data move operations is an ALU or MAC data register—an XOP register for the DM bus access and a YOP register for the PM bus access.

XOP register: $AX0, AX1, MX0, \text{ or } MX1$

YOP register: $AY0, AY1, MY0, \text{ or } MY1$

STATUS FLAGS

The status flags generated as a result of the computation depends on the compute operation the instruction performs. For more information, see the status flags section of the computation’s reference page.

DETAILS

The memory read operations use register indirect addressing with post-modify ($I_{reg} += M_{reg}$). For linear indirect addressing, you must initialize the L_x register of the corresponding I_{reg} register to 0. For circular indirect addressing, you must set the buffer's length and base address with the corresponding L_{reg} and B_{reg} registers. For more information on addressing, see the *ADSP-219x/2191 DSP Hardware Reference*.

The $DM()$ reference uses the 16-bit DM data bus, and the $PM()$ reference uses the 24-bit PM data bus. For PM data moves, the destination data register receives the sixteen MSBs from 24-bit memory, and the PX register catches the eight LSBs. To use all twenty-four bits of the memory data, you must transfer the eight LSBs from PX to another data register. Otherwise, the eight LSBs will be lost.

The address of the access, not the $PM()$ or $DM()$ reference, selects the memory bank. So, the DM reference could access 24-bit memory, and the PM reference could access 16-bit memory. DM reads of 24-bit memory result in the specified data register receiving bits 23:8 from memory. PM reads of 16-bit memory result in the specified data register receiving bits 23:8 from memory. When the PX register is loaded using a 16-bit memory access (PM reference to 16-bit memory or DM reference to 24-bit memory), the DSP clears (=0) the 8-LSBs of PX .

This multifunction instruction requires the DSP to fetch three items from memory: the instruction and two data words. The number of cycles required to execute it depends on whether the instruction generates bus conflicts:

Execution	Conditions
1 cycle	If the instruction is already cached and the data are from different memory banks

Multifunction Instructions

Execution	Conditions
2 cycles	If only one bus conflict occurs—data vs. data or instruction vs. data
3 cycles	If two bus conflicts occur—instruction vs. data vs. data

EXAMPLES

```
AR = AX0 - AY0,  
MX1 = DM(I3 += M0),  
MY1 = PM(I5 += M4);           /* sub and dual read */
```

```
AR = AX0 + AY0,  
MX0 = DM(I1 += M0),  
MY0 = PM(I4 += M4);           /* add and dual read */
```

```
MR = MX0 * MY0 (SS),  
MX0 = DM(I2 += M2),  
MY0 = PM(I7 += M7);           /* mult and dual read */
```

SEE ALSO

- [“Type 1: Compute | DregX«...DM | DregY«...PM” on page 9-21](#)
- [“ALU Instructions” on page 3-1.](#)
- [“MAC Instructions” on page 4-1.](#)
- [“Arithmetic Status \(ASTAT\) Register” on page 2-5](#)

Dual Memory Read

$$\left(\begin{array}{c} AX0 \\ AX1 \\ MX0 \\ MX1 \end{array} \right) = DM(\left(\begin{array}{c} I0 \\ I1 \\ I2 \\ I3 \end{array} \right) += \left(\begin{array}{c} M0 \\ M1 \\ M2 \\ M3 \end{array} \right)), \left(\begin{array}{c} AY0 \\ AY1 \\ MY0 \\ MY1 \end{array} \right) = PM(\left(\begin{array}{c} I4 \\ I5 \\ I6 \\ I7 \end{array} \right) += \left(\begin{array}{c} M4 \\ M5 \\ M6 \\ M7 \end{array} \right));$$

FUNCTION


Performs two memory read operations, one over the 16-bit DM data bus and the other over the 24-bit PM data bus.

Each read operation moves the contents of the specified memory location to its respective destination register. The destination of both memory read operations is an ALU or MAC data register. The DM bus read loads an ALU or MAC DREG_x register, and the PM bus read loads an ALU or MAC DREG_y register. The memory data is always right-justified in the destination data register.

INPUT

Both data move operations use two DAG registers, index (*Ireg*) and modify (*Mreg*), to generate memory addresses—DAG1 registers for the DM bus access, and DAG2 registers for the PM bus access. For details on DAG registers and data addressing, see [“Data Move Instructions” on page 7-1](#).

- DM/DAG1 I0, I1, I2, or I3 (index registers)
 M0, M1, M2, or M3 (modify registers)
- PM/DAG2 I4, I5, I6, or I7 (index registers)
 M4, M5, M6, or M7 (modify registers)

 You can use any index register with any modify register from the same DAG. You cannot pair a DAG1 register with a DAG2 register.

Multifunction Instructions

OUTPUT

The destination register for both data move operations is an ALU or MAC data register—an XOP register for the DM bus access and a YOP register for the PM bus access.

XOP AX0, AX1, MX0, or MX1

YOP AY0, AY1, MY0, or MY1

STATUS FLAGS

None affected.

DETAILS

The memory read operations use register indirect addressing with post-modify ($I_{reg} += M_{reg}$). For linear indirect addressing, you must initialize the Lreg register of the corresponding Ireg register to 0. For circular indirect addressing, you must set the buffer's length and base address with the corresponding Lreg and Breg registers. For more information on addressing, see the *ADSP-219x/2191 DSP Hardware Reference*.

The DM reference uses the 16-bit DM data bus, and the PM reference uses the 24-bit PM data bus. For PM data moves, the destination data register receives the sixteen MSBs from 24-bit memory, and the PX register catches the eight LSBs. To use all twenty-four bits of the memory data, you must transfer the eight LSBs from PX to another data register. Otherwise, the eight LSBs will be lost.

The address of the access, not the PM() or DM() reference, selects the memory bank. So, the DM() reference could access 24-bit memory, and the PM() reference could access 16-bit memory. DM reads of 24-bit memory result in the specified data register receiving bits 23:8 from memory. PM reads of 16-bit memory result in the specified data register receiving bits 23:8 from memory. When the PX register is loaded using a 16-bit memory access (PM reference to 16-bit memory or DM reference to 24-bit memory), the DSP clears (=0) the 8-LSBs of PX.

Dual Memory Read

This multifunction instruction requires the DSP to fetch three items from memory: the instruction and two data words. The number of cycles required to execute it depends on whether the instruction generates bus conflicts:

Execution	Conditions
1 cycle	If the instruction is already cached and the data are from different memory banks
2 cycles	If only one bus conflict occurs—data vs. data or instruction vs. data
3 cycles	If two bus conflicts occur—instruction vs. data vs. data

EXAMPLES

```
MX0 = DM(I0 += M0),  
MY0 = PM(I5 += M4);           /* dual read */  
  
AX1 = DM(I3 += M0),  
AY1 = PM(I6 += M4);           /* dual read */
```

SEE ALSO

- [“Type 1: Compute | DregX«...DM | DregY«...PM” on page 9-21](#)

Multifunction Instructions

Compute with Memory Read

<ALU>	, DREG =	DM	(I0	+=	M0) ;
<MAC>		PM		I1		M1	
<SHIFT>				I2		M2	
				I3		M3	
				I4		M4	
				I5		M5	
				I6		M6	
				I7		M7	

FUNCTION

Combines an ALU, MAC, or shifter operation with a 16-bit read from memory over the DM (data memory) bus. The restricted register forms—*using XOP and YOP registers, not the DREG register file*—of all ALU, MAC, or Shifter instructions are supported, except for the MAC saturate instruction, the divide primitives `DIVS` and `DIVQ`, and Shift immediate. Also, the multifunction ALU, MAC, and Shifter instructions may not use conditional (IF) options.

The compute operation executes first, using the current contents of the data registers as input operands. Then the memory read operation executes, overwriting the contents of the destination data register with new data from memory.


The read operation moves the contents of the memory location to the specified destination register. The destination of the memory read operation is an ALU, MAC, or shifter data register. The memory data is always right-justified in the destination data register.

INPUT

Valid input operands for the compute depend on the operation's computation unit. For more information, see the input descriptions in [“ALU Instructions” on page 3-1](#), [“MAC Instructions” on page 4-1](#), and [“Shifter Instructions” on page 5-1](#).

The data move operation uses two DAG registers, index (I_{reg}) and modify (M_{reg}), to generate memory addresses. Regardless of which DAG registers are used, all accesses occur over the DM bus. For details on DAG registers and data addressing, see “Data Move Instructions” on page 7-1.

- DAG1 $I0, I1, I2, \text{ or } I3$ (index registers)
 $M0, M1, M2, \text{ or } M3$ (modify registers)
- DAG2 $I4, I5, I6, \text{ or } I7$ (index registers)
 $M4, M5, M6, \text{ or } M7$ (modify registers)

 You can use any index register with any modify register from the same DAG. You cannot pair a DAG1 register with a DAG2 register.

OUTPUT

The result register for the compute operation is always the computation unit’s result or feedback register.

AR/AF ALU operations

MR/SR MAC operations

SR/SE Shifter operations

The destination register for the data move operation is any register file data register. You can use any of these data registers for the DREG destination:

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

STATUS FLAGS

The status flags generated as a result of the computation depends on the compute operation the instruction performs. For more information, see the status flags section of the computation’s reference page.

Multifunction Instructions

DETAILS

The memory read operation uses indirect addressing with postmodify ($I_{reg} += M_{reg}$) and always accesses 16-bit data over the DM bus. For linear indirect addressing, you must initialize the L_x register of the corresponding I_{reg} register to 0. For circular indirect addressing, you must set the buffer's length and base address with the corresponding L_{reg} and B_{reg} registers. For more information on addressing, see the *ADSP-219x/2191 DSP Hardware Reference*.

Since the data accesses occur over the DM data bus only, the $PM()$ and $DM()$ references are semantically identical in this instruction. The address of the access selects the memory bank, so this instruction could access 24-bit memory. If so, the specified data register receives bits 23:8 from memory. Since the $PM()$ reference does not activate the PM data bus, the PX register is not filled with any data.

This multifunction instruction requires the DSP to fetch two items from memory: the instruction and one data word. The number of cycles required to execute this instruction depends on whether it generates a bus conflict:

Execution	Conditions
1 cycle	If no bus conflict occurs.
2 cycles	If an instruction vs. data conflict occurs on the bus

EXAMPLES

```
AR = AX0 - AY1 + C - 1,
AX0 = DM(I1 += M0);           /* ALU operation and mem read */

MR = MX1 * MY0 (SS),
SR1 = PM(I4 += M4);         /* MAC operation and mem read */

AR = 3; SE = AR;           /* shift code, lshift 3 bits */
```

Compute with Memory Read

```
SI = 0xB6A3;                /* value of hi word of input */
SR = ASHIFT SI (HI),        /* ashift hi word and mem read */
SI = DM(I0 += M0);

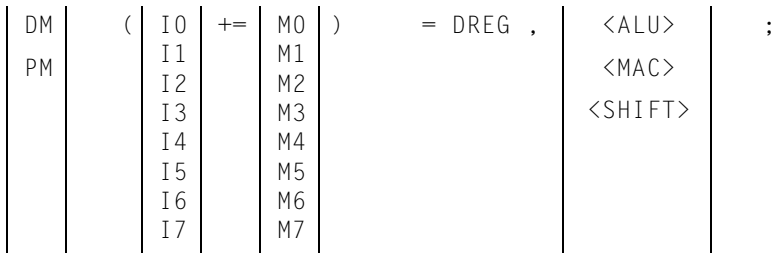
AR = 3; SE = AR;           /* shift code lshift 3 bits */
SI = 0x765D;               /* value of lo word of input */
SR = SR OR LSHIFT SI (L0), /* lshift lo word and mem read */
SI = DM(I0 += M0);
```

SEE ALSO

- [“Type 4: Compute | Dreg «...» DM/PM” on page 9-23](#)
- [“Type 12: Shift | Dreg «...» DM/PM” on page 9-35](#)
- [“ALU Instructions” on page 3-1](#)
- [“MAC Instructions” on page 4-1](#)
- [“Shifter Instructions” on page 5-1](#)
- [“Arithmetic Status \(ASTAT\) Register” on page 2-5](#)

Multifunction Instructions

Compute with Memory Write



FUNCTION

Combines an ALU, MAC, or shifter operation with a 16-bit write to memory over the DM (data memory) bus. The restricted register forms—*using XOP and YOP registers, not the DREG register file*—of all ALU, MAC, and Shifter instructions are supported, except for the MAC saturate instruction, the divide primitives `DIVS` and `DIVQ`, and Shift immediate. Also, the multifunction ALU, MAC, and Shifter instructions may not use conditional (`IF`) options.

The write operation executes first, transferring the current contents of the data register to the specified memory location. Then the compute operation executes, overwriting the contents of the destination data register with the result.


The source of data for the memory write operation is an ALU, MAC, or shifter result or feedback register. The data is always right-justified in the destination memory location.

INPUT

Valid input operands for the compute depend on the operation's computation unit. For more information, see the input descriptions in [“ALU Instructions” on page 3-1](#), [“MAC Instructions” on page 4-1](#), and [“Shifter Instructions” on page 5-1](#).

The data move operation uses two DAG registers, index (I_{reg}) and modify (M_{reg}), to generate memory addresses. Regardless of which DAG registers are used, all accesses occur over the DM bus. For details on DAG registers and data addressing, see “Data Move Instructions” on page 7-1.

- DAG1 $I0, I1, I2, \text{ or } I3$ (index registers)
 $M0, M1, M2, \text{ or } M3$ (modify registers)
- DAG2 $I4, I5, I6, \text{ or } I7$ (index registers)
 $M4, M5, M6, \text{ or } M7$ (modify registers)

 You can use any index register with any modify register from the same DAG. You cannot pair a DAG1 register with a DAG2 register.

OUTPUT

The destination register for the compute operation is always the computation unit’s result or feedback register.

AR/AF	ALU operations
MR/SR	MAC operations
SR/SE	Shifter operations

The source register for the data move operation is any register file data register. You can use any of these data registers for the $DREG$ source:

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

STATUS FLAGS

The status flags generated as a result of the computation depends on the compute operation the instruction performs. For more information, see the status flags section of the computation’s reference page.

Multifunction Instructions

DETAILS

The memory write operation uses indirect addressing with postmodify ($Ireg += Mreg$) and always transfers 16-bit data over the DM bus. For linear indirect addressing, you must initialize the Lreg register of the corresponding Ireg register to 0. For circular indirect addressing, you must set the buffer's length and base address with the corresponding Lreg and Breg registers. For more information on addressing, see the *ADSP-219x/2191 DSP Hardware Reference*.

Since transfers occur over the DM data bus only, the PM() and DM() references are semantically identical in this instruction. The address of the access selects the memory bank, so this instruction could access 24-bit memory. If so, the operation writes bits 15:0 from the specified data register to bits 23:8 of the specified memory location. Since the PM() reference does not activate the PM data bus, the PX register does not supply any data.

This multifunction instruction requires the DSP to fetch one item from memory and write one item to memory: the instruction and one data word. The number of cycles required to execute the instruction depends on whether it generates a bus conflict:

Execution	Conditions
1 cycle	If no bus conflict occurs.
2 cycles	If an instruction vs. data conflict occurs on the bus

Except for SR2, you can use the same data register in both the compute and memory write operations—as the result register for the computation and as the source register for the data move operation.

EXAMPLES

```
DM(I1 += M0) = AX0,  
AR = AX0 - AY1 + C - 1;    /* mem write and ALU operation */
```

Compute with Memory Write

```
PM(I4 += M4) = SR1,  
MR = MX1 * MY0 (SS);      /* mem write and MAC operation */  
  
AR = 3; SE = AR;          /* shift code, lshift 3 bits */  
SI = 0xB6A3;              /* value of hi word of input */  
DM(I0 += M0) = SI,  
SR = ASHIFT SI (HI);      /* mem write and ashift hi word */  
  
AR = 3; SE = AR;          /* shift code lshift 3 bits */  
SI = 0x765D;              /* value of lo word of input */  
DM(I0 += M0) = SI,  
SR = SR OR LSHIFT SI (L0); /* mem write and lshift lo word */
```

SEE ALSO

- [“Type 4: Compute | Dreg «…» DM/PM” on page 9-23](#)
- [“Type 12: Shift | Dreg «…» DM/PM” on page 9-35](#)
- [“ALU Instructions” on page 3-1](#)
- [“MAC Instructions” on page 4-1](#)
- [“Shifter Instructions” on page 5-1](#)
- [“Arithmetic Status \(ASTAT\) Register” on page 2-5](#)

Multifunction Instructions

Compute with Register to Register Move

<p><ALU></p> <p><MAC></p> <p><SHIFT></p>	<p>, DREG1 = DREG2 ;</p>
--	--------------------------

FUNCTION

Combines an ALU, MAC, or shifter operation with a register to register move. The restricted register forms—*using XOP and YOP registers, not the DREG register file*—of all ALU, MAC, and Shifter instructions are supported, except for the MAC saturate instruction, the divide primitives DIVS and DIVQ, and Shift immediate. Also, the multifunction ALU, MAC, and Shifter instructions may not use conditional (IF) options.

The compute operation executes first, using the current contents of the data register. Then the data move executes, overwriting the contents of the destination data register with the contents of the source register.

The source and destination of the data move operation is an ALU, MAC, or shifter data register. The transferred data is always right-justified in the destination data register.

INPUT

Valid input operands for the compute depend on the operation's computation unit. For more information, see the input descriptions in [“ALU Instructions” on page 3-1](#), [“MAC Instructions” on page 4-1](#), and [“Shifter Instructions” on page 5-1](#).

Compute with Register to Register Move

OUTPUT

The result register for the compute operation is always the computation unit's result or feedback register.

AR/AF	ALU operations
MR/SR	MAC operations
SR/SE	Shifter operations

The source (DREG2) and destination (DREG1) registers for the data move operation are any register file data registers. You can use any of these data registers for the DREG source and destination:

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

STATUS FLAGS

The status flags generated as a result of the computation depends on the compute operation the instruction performs. For more information, see the status flags section of the computation's reference page.

DETAILS

Except for SR2, you can use the same data register in both the compute and data move operations—as the result register for the computation and as the source register for the data move operation.

If you use AR as the source and destination in the data move operation (AR = AR), the compute operation generates status only—no computation results. For more information, see [“Generate ALU Status Only: NONE” on page 3-44](#).

Multifunction Instructions

EXAMPLES

```
AR = AX1 + AY1, MX0 = AR;          /* add and reg.-to-reg. move */
MR = MX1 * MY0 (US), MY0 = AR;    /* mult and reg.-to-reg. move */
AR = 3; SE = AR;                  /* shift code, lshift 3 bits */
SI = 0xB6A3;                      /* value of hi word of input */
SR = ASHIFT SI (HI),              /* ashift hi word reg move */
SI = MR0;
```

SEE ALSO

- [“Type 8: Compute | Dreg1 «... Dreg2” on page 9-26](#)
- [“Type 14: Shift | Dreg1 «... Dreg2” on page 9-36](#)
- [“ALU Instructions” on page 3-1](#)
- [“MAC Instructions” on page 4-1](#)
- [“Shifter Instructions” on page 5-1](#)
- [“Arithmetic Status \(ASTAT\) Register” on page 2-5](#)