

Manual Update Sheet

DATE: June 4, 2004

Document Being Updated: *TMS320C6000 CPU and Instruction Set Reference Guide*

Literature Number Being Updated: SPRU189F

This Manual Update Sheet (SPRZ168E) describes changes for the *TMS320C6000 CPU and Instruction Set Reference Guide* (SPRU189F).

Updates within paragraphs, figures, and tables appear in a **bold typeface**.

New updates within this revision of the manual update sheet show their page number in a **bold typeface**.

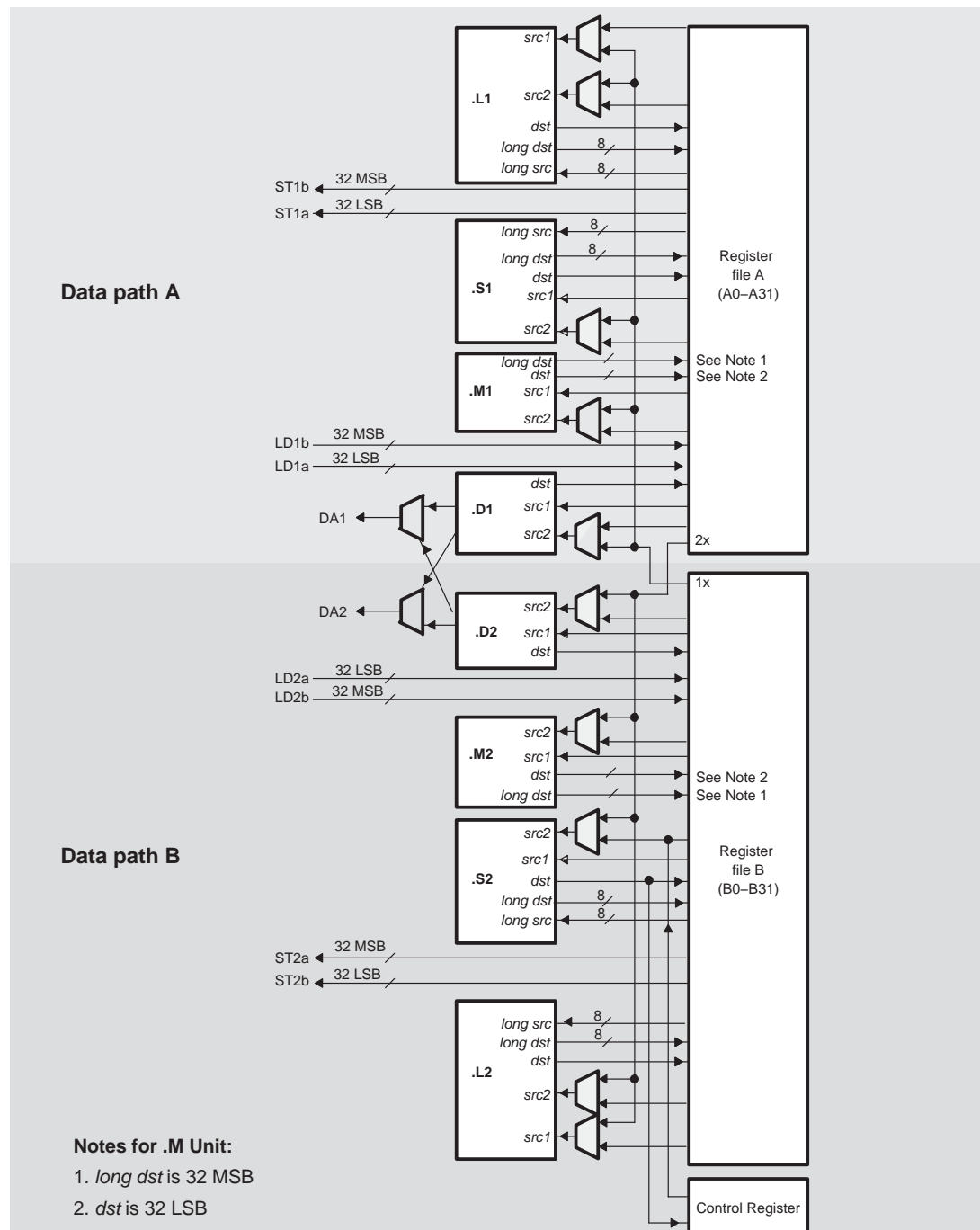
Page:	Change or Add:
--------------	-----------------------

2–4	Change Figure 2–3 in data paths A and B, the connections were changed for long dst, src, and long src.
-----	--

In the C64x, the long source (src) data paths on the .L and .S units are no longer shared with the memory store paths (ST1/ST2). Similarly, the long destination (dst) paths are no longer shared with the memory load paths (LD1/LD2). Each data path is now independently connected to the register file. This removes all of the long (40-bit) operand/destination restrictions on the C64x, as compared to the C62x/C67x.

Page: Change or Add:

Figure 2–3. TMS320C64x CPU Data Path



Page: **Change or Add:**

2–18 Change the Function of the CPU ID (bits 31–24) in Table 2–6.

Change the Function of the Revision ID (bits 23–16) in Table 2–6.

Change the footnote in Table 2–6.

Table 2–6. Control Status Register Field Descriptions

Bit Position	Width	Field Name	Function
31-24	8	CPU ID and Revision ID	
		CPU ID	CPU ID identifies which of these CPUs: CPU ID = 00h: indicates C62x CPU ID= 02h: indicates C67x CPU ID = 0Ch: indicates C64x and DM64x
23-16	8	Revision ID	Identifies silicon revision of the CPU. For the most current silicon revision information, see the device-specific datasheet.
15-10	6	PWRD	Control power-down modes; the values are always read as zero. [†]
9	1	SAT	The saturate bit, set when any unit performs a saturate, can be cleared only by the MVC instruction and can be set only by a functional unit. The set by a functional unit has priority over a clear (by the MVC instruction) if they occur on the same cycle. The saturate bit is set one full cycle (one delay slot) after a saturate occurs. This bit will not be modified by a conditional instruction whose condition is false.
8	1	EN	Endian bit: 1 = little endian, 0 = big endian [†]
7-5	3	PCC	Program cache control mode [†]
4-2	3	DCC	Data cache control mode [†]
1	1	PGIE	Previous GIE (global interrupt enable); saves GIE when an interrupt is taken.
0	1	GIE	Global interrupt enable; enables (1) or disables (0) all interrupts except the reset interrupt and NMI (nonmaskable interrupt).

[†] See the *Two-Level Internal Memory Reference Guide* (SPRU609 or SPRU610) for more information.

Page: Change or Add:

3–17 Add the following paragraphs to the end of section 3.7.1:

Two instructions using the same functional unit cannot write their results in the same instruction cycle. For example, the following code is invalid:

```
LDW      .D1      *A10, A6
NOP                      3
MV       .D1      A4, A2
```

The **LDW** instruction has 4 delay slots and the **MV** instruction has zero delay slots. They will try to write their calculation results in the same instruction cycle (immediately after the **MV** instruction). Since the .D1 unit has only a single write port to the register file, a resource conflict results.

3–30 Add two rows for C64x only and the Mnemonic column to the ADD(U) instruction. The ADD instruction on the C64x can also use cross-path operands on the .D units as well as on the .S and .L units.

ADD(U)
Signed or Unsigned Integer Addition Without Saturation
Syntax

ADD (.unit) *src1*, *src2*, *dst*
 or
ADDU (.L1 or .L2) *src1*, *src2*, *dst*
 or
ADD (.D1 or .D2) *src2*, *src1*, *dst*
 .unit = .L1, .L2, .S1, .S2

Opcode map field used...	For operand type...	Unit	Opfield	Mnemonic
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint sint	.L1, .L2	0000011	ADD
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint slong	.L1, .L2	0100011	ADD
<i>src1</i> <i>src2</i> <i>dst</i>	uint xuint ulong	.L1, .L2	0101011	ADDU
<i>src1</i> <i>src2</i> <i>dst</i>	xsint slong slong	.L1, .L2	0100001	ADD
<i>src1</i> <i>src2</i> <i>dst</i>	xuint ulong ulong	.L1, .L2	0101001	ADDU
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 xsint sint	.L1, .L2	0000010	ADD

Page: Change or Add:

Opcode map field used...	For operand type...	Unit	Opfield	Mnemonic
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 slong slong	.L1, .L2	0100000	ADD
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint sint	.S1, .S2	000111	ADD
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 xsint sint	.S1, .S2	000110	ADD
<i>src2</i> <i>src1</i> <i>dst</i>	sint sint sint	.D1, .D2	010000	ADD
<i>src2</i> <i>src1</i> <i>dst</i>	sint ucst5 sint	.D1, .D2	010010	ADD
C64x only:				
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint sint	.D1, .D2	101010	ADD
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 xsint sint	.D1, .D2	101011	ADD

Page: **Change or Add:**

3–105 Add a row for C64x only to the SADD instruction. The SADD instruction on the C64x can also be performed on the .S units as well as on the .L units.

SADD

Integer Addition With Saturation to Result Size

Syntax

SADD (.unit) *src1*, *src2*, *dst*

.unit = .L1, .L2, .S1, .S2

Opcode map field used...	For operand type...	Unit	Opfield
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint sint	.L1, .L2	0010011
<i>src1</i> <i>src2</i> <i>dst</i>	xsint slong slong	.L1, .L2	0110001
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 xsint sint	.L1, .L2	0010010
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 slong slong	.L1, .L2	0110000
C64x only:			
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint sint	.S1, .S2	1000001

Page: **Change or Add:**

3–132

Add two rows for C64x only to the SUB(U) instruction. The SUB instruction on the C64x can also use cross-path operands on the .D units as well as on the .S and .L units and the C64x also provides a subtract instruction of the form $\text{src2} - \text{src1}$ on the .S unit.

SUB(U)*Signed or Unsigned Integer Subtraction Without Saturation***Syntax****SUB** (.unit) *src1*, *src2*, *dst*

or

SUBU (.unit) *src1*, *src2*, *dst*

or

SUB (.D1 or .D2) *src2*, *src1*, *dst*

.unit = .L1, .L2, .S1, .S2, .D1, .D2

Opcode map field used...	For operand type...	Unit	Opfield	Mnemonic
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint sint	.L1, .L2	0000111	SUB
<i>src1</i> <i>src2</i> <i>dst</i>	xsint sint sint	.L1, .L2	0010111	SUB
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint slong	.L1, .L2	0100111	SUB
<i>src1</i> <i>src2</i> <i>dst</i>	xsint sint slong	.L1, .L2	0110111	SUB
<i>src1</i> <i>src2</i> <i>dst</i>	uint xuint ulong	.L1, .L2	0101111	SUBU
<i>src1</i> <i>src2</i> <i>dst</i>	xuint uint ulong	.L1, .L2	0111111	SUBU
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 xsint sint	.L1, .L2	0000110	SUB
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 slong slong	.L1, .L2	0100100	SUB
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint sint	.S1, .S2	010111	SUB
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 xsint sint	.S1, .S2	010110	SUB

Page:

Change or Add:

Opcode map field used...	For operand type...	Unit	Opfield	Mnemonic
<i>src2</i> <i>src1</i> <i>dst</i>	sint sint sint	.D1, .D2	010001	SUB
<i>src2</i> <i>src1</i> <i>dst</i>	sint ucst5 sint	.D1, .D2	010011	SUB
C64x only:				
<i>src2</i> <i>src1</i> <i>dst</i>	xsint sint sint	.S1, .S2	110101	SUB src2 – src1
<i>src1</i> <i>src2</i> <i>dst</i>	sint xsint sint	.D1, .D2	101100	SUB src1 – src2

Page: **Change or Add:**

3–119 Change the src2 operand type for the SMPY mnemonic in the SMPY(HL/LH/H) instruction:

Opcode map field used...	For operand type...	Unit	Opfield	Mnemonic
<i>src1</i> <i>src2</i> <i>dst</i>	slsb16 xslsb16 sint	.M1, .M2	11010	SMPY
<i>src1</i> <i>src2</i> <i>dst</i>	smsb16 xslsb16 sint	.M1, .M2	01010	SMPYHL
<i>src1</i> <i>src2</i> <i>dst</i>	slsb16 xsmsb16 sint	.M1, .M2	10010	SMPYLH
<i>src1</i> <i>src2</i> <i>dst</i>	smsb16 xsmsb16 sint	.M1, .M2	00010	SMPYH

3–140 Change the syntax of the XOR instruction:

Syntax **XOR** (.unit) *src1*, *src2*, *dst*

Page: **Change or Add:**

5–16 Change Table 5–7 and the paragraphs that follow the table in section 5.6.4:

Table 5–7. Constraint Differences Between C62x/C67x and C64x Registers

Situation	C62x	C67x	C64x
Both .S and .L units on the same side use a long source.	Conflict	Conflict	OK
Both .S and .L units on the same side write a long result.	Conflict	Conflict	OK
A store reads the data to store from the same side that the .S unit reads a long source.	Conflict	Conflict	OK
A store reads the data to store from the same side that the .L unit reads a long source.	Conflict	Conflict	OK
An LDDW writes back data (loaded from memory) to the same side that the .S unit writes a long result.	N/A	Conflict	OK
An LDDW writes back data (loaded from memory) to the same side that the .L unit writes a long result.	N/A	Conflict	OK

Legend: Conflict = Conflict exists
 OK = No conflict exists
 N/A = Not Applicable; instruction not available to core

On the C62x and C67x, only one long result may be issued per register file in an execute packet. **On the C64x, the .L, .S, and .D units can operate independently.** Up to two instructions with long results may be issued per side in an execute packet. **On the C67x, double word load instructions conflict with long results from the .S units. On the C62x/C67x, all stores conflict with a long source on the .S unit. On the C64x, there are no conflicts using long sources and destinations on the .L and .S units while loading/storing data with the .D unit.**

The following execute packet is **invalid on the C62x/C67x cores**, because the .D unit store on the T1 path conflicts with the long source on the .S1 unit:

```
ADD .S1    A1,A5:A4, A3:A2 ; \ Long source on .S unit and a
|| STW .D1T1 A8,*A9        ; / store on the T1 path of the .D unit
```

Note: This execute packet is valid on the C64x core.

The following code sequence is **invalid on the C67x core and valid on the C64x core**. This example is not applicable to the C62x core because it uses the **LDDW** instruction that is not supported on the C62x core.

```
LDDW .D1T1 *A16,A11:A10 ; \ Double word load written to
                        ; A11:A10 on .D1
NOP 3                    ; conflicts after 3 cycles
SHL .S1    A8,A9,A7:A6 ; / with write to A7:A6 on .S1
```

Page: Change or Add:

5–17 Change the second paragraph in section 5.6.4:

The following code sequences are invalid on the C62x and C67x cores but valid on the C64x core:

```

      ADD .L1  A1,A5:A4,A3:A2    ; \ Two long writes
|| SHL .S1  A8,A9,A7:A6        ; / on A register file

      LDW .D1T1 *A13,A11        ; \ Word read on T1 path of .D1
      NOP 3                     ; / does not conflict after
      ADD .L1  A4,A1,A3:A2      ; \ 3 cycles with long write on
|| SHL .S1  A8,A2,A7:A6        ; / .L1 unit and long write on
                                   ; .S1 unit

      ADD .L1  A1,A5:A4,A3:A2    ; \ Long source operand on .L1
|| STW .D1T1  A8,*A9           ; does not conflict with
                                   ; / store word on the T1 path
                                   ; of .D1

      SHR .S1  A5:A4, A1, A3:A2  ; \ No long read on .S1
|| STW .D1T1  A8,*A9           ; / with the store on T1 path
                                   ; of .D1

```

5–17 Delete the fourth paragraph in section 5.6.4:

The following execute packet is invalid on all cores:

```

      SHR .S1  A5:A4, A1, A3:A2 ; \ No long read on .S1 with
|| STW .D1T1  A8,*A9          ; / the store on T1 path of .D1

```

5–49 Change the operand types in the BTR instruction:

Opcode map field used...	For operand type...	Unit	Opfield
<i>src2</i>	xuint	.M1, .M2	11111
<i>dst</i>	uint		

5–88 Change the Note in the DOTPNRSU2 instruction:

Note:

Certain combinations of operands for the DOTPNRSU2 instruction will result in an overflow condition. If an overflow does occur, the result is undefined. Overflow can be avoided if the difference of the two products plus the rounding term is less than or equal to $2^{31} - 1$ for a positive sum and greater than or equal to -2^{31} for a negative sum.

Page: **Change or Add:**

5–89 Add Example 3 in the DOTPNRSU2 instruction:

Example 3 DOTPNRSU2 .M2 B12, B23, B11;

Before instruction			4 cycles after instruction		
B12	7FFF 8000h	32767 -32768 signed	B12	7FFF 8000h	32767 -32768 signed
B23	FFFF FFFFh	65535 65535 unsigned	B23	FFFF FFFFh	65535 65535 unsigned
B11	XXXX XXXXh		B11	XXXX XXXXh	Overflow occurs; result undefined

5–93 Change the Note in the DOTPRSU2 instruction:

Note:

Certain combinations of operands for the DOTPRSU2 instruction will result in an overflow condition. If an overflow does occur, the result is undefined. Overflow can be avoided if the sum of the two products plus the rounding term is less than or equal to $2^{31} - 1$ for a positive sum and greater than or equal to -2^{31} for a negative sum.

5–94 Add Example 3 in the DOTPRSU2 instruction:

Example 3 DOTPRSU2 .M2 B12, B23, B11;

Before instruction			4 cycles after instruction		
B12	7FFF 7FFFh	32767 32767 signed	B12	7FFF 7FFFh	32767 32767 signed
B23	FFFF FFFFh	65535 65535 unsigned	B23	FFFF FFFFh	65535 65535 unsigned
B11	XXXX XXXXh		B11	XXXX XXXXh	Overflow occurs; result undefined

Page: Change or Add:

5–107 Change the second paragraph in the LDDW instruction:

Both *offsetR* and *baseR* must be in the same register file and on the same side as the .D unit used. The *y* bit in the opcode determines the .D unit and the register file used: *y* = 0 selects the .D1 unit and the *baseR* and *offsetR* from the A register file, and *y* = 1 selects the .D2 unit and *baseR* and *offsetR* from the B register file. The *s* bit determines the register file into which the *dst* is loaded: *s* = 0 indicates that *dst* is in the A register file, and *s* = 1 indicates that *dst* is in the B register file. **The *r* bit has a value of 1 for the LDDW instruction.** The *dst* field must always be an even value because LDDW loads register pairs. Therefore, bit 23 is always zero. Furthermore, the value of the *ld/st* field is 110.

5–111 Change the second paragraph in the LDNDW instruction:

The *dst* can be in either register file, regardless of the .D unit or *baseR* or *offsetR* used. The *s* bit determines which file *dst* will be loaded into: *s* = 0 indicates *dst* will be in the A register file and *s* = 1 indicates *dst* will be loaded in the B register file. **The *r* bit has a value of 1 for the LDNDW instruction. Furthermore, the value of the *ld/st* field is 010.**

5–115 Change the first paragraph in the LDNW instruction:

The *dst* can be in either register file, regardless of the .D unit or *baseR* or *offsetR* used. The *s* bit determines which file *dst* will be loaded into: *s* = 0 indicates *dst* will be in the A register file and *s* = 1 indicates *dst* will be loaded in the B register file. **The *r* bit is always 1.**

5–151 Change the *src1* operand type in the MPYU4 instruction:

Opcode map field used...	For operand type...	Unit	Opfield
<i>src1</i>	u4	.M1, .M2	00100
<i>src2</i>	xu4		
<i>dst</i>	dwu4		

Page: Change or Add:

5–159 Change the opcode map field for the first operand type in the OR instruction:

Opcode map field used...	For operand type...	Unit	Opfield
<i>src1</i> <i>src2</i> <i>dst</i>	uint xuint uint	.D1, .D2	0001
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 xuint uint	.D1, .D2	0011
<i>src1</i> <i>src2</i> <i>dst</i>	uint xint uint	.L1, .L2	1111111
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 xuint uint	.L1, .L2	1111110
<i>src1</i> <i>src2</i> <i>dst</i>	uint xunit uint	.S1, .S2	011011
<i>src1</i> <i>src2</i> <i>dst</i>	scst5 xuint uint	.S1, .S2	011010

5–208 Change the dst operand type in the SMPY2 instruction:

Opcode map field used...	For operand type...	Unit	Opfield
<i>src1</i> <i>src2</i> <i>dst</i>	s2 xs2 sllong	.M1, .M2	00001

5–226 Change the first paragraph in the STDW instruction:

The *srcd* pair can be in either register file, regardless of the .D unit or baseR or offsetR used. The s bit determines which file *srcd* will be loaded from: s = 0 indicates *srcd* will be in the A register file and s = 1 indicates *srcd* will be in the B register file. **The r bit is always 1.**

5–230 Change the first paragraph in the STNDW instruction:

The *srcd* pair can be in either register file, regardless of the .D unit or baseR or offsetR used. The s bit determines which file *srcd* will be loaded from: s = 0 indicates *srcd* will be in the A register file and s = 1 indicates *srcd* will be in the B register file. **The r bit is always 1.**

Page: **Change or Add:**


5–234 Change the first paragraph in the STNW instruction:

The *src* can be in either register file, regardless of the .D unit or baseR or offsetR used. The *s* bit determines which file *src* will be loaded from: *s* = 0 indicates *src* will be in the A register file and *s* = 1 indicates *src* will be in the B register file.
The *r* bit is always 1.

7–27 Change the Instruction Execution in Cycles 2–4 for Load and Store in Table 7–9:

 Table 7–9. *MPYI .M-Unit Instruction Constraints*

Instruction Execution										
Cycle	1	2	3	4	5	6	7	8	9	10
MPYI	R	R	R	R					W	
Instruction Type	Subsequent Same-Unit Instruction Executable									
16 × 16 multiply		Xr	Xr	Xr	✓	✓	✓	Xw	✓	✓
4-cycle		Xr	Xr	Xr	Xu	Xw	Xu	✓	✓	✓
MPYI		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓
MPYID		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓
MPYDP		Xr	Xr	Xr	Xu	Xu	Xu	✓	✓	✓
Instruction Type	Same Side, Different Unit, Both Using Cross Path Executable									
Single-cycle		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓
Load		✓	✓	✓	✓	✓	✓	✓	✓	✓
Store		✓	✓	✓	✓	✓	✓	✓	✓	✓
DP compare		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓
2-cycle DP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓
Branch		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓
4-cycle		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓
INTDP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓
ADDDP/SUBDP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓


Legend:  E1 phase of the single-cycle instruction
 R Sources read for the instruction
 W Destinations written for the instruction
 ✓ Next instruction can enter E1 during cycle
 Xr Next instruction cannot enter E1 during cycle–read/decode constraint
 Xw Next instruction cannot enter E1 during cycle–write constraint
 Xu Next instruction cannot enter E1 during cycle–other resource conflict

Page: **Change or Add:**

7–28 Change the Instruction Execution in Cycles 2–4 for Load and Store in Table 7–10:

Table 7–10. MPYID .M-Unit Instruction Constraints

Instruction Execution											
Cycle	1	2	3	4	5	6	7	8	9	10	11
MPYID	R	R	R	R					W	W	
Instruction Type	Subsequent Same-Unit Instruction Executable										
16 × 16 multiply		Xr	Xr	Xr	✓	✓	✓	Xw	Xw	✓	✓
4-cycle		Xr	Xr	Xr	Xu	Xw	Xw	✓	✓	✓	✓
MPYI		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
MPYID		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
MPYDP		Xr	Xr	Xr	Xu	Xu	Xu	✓	✓	✓	✓
Instruction Type	Same Side, Different Unit, Both Using Cross Path Executable										
Single-cycle		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
Load		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Store		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DP compare		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
2-cycle DP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
Branch		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
4-cycle		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
INTDP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
ADDDP/SUBDP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓


Legend:  E1 phase of the single-cycle instruction
 R Sources read for the instruction
 W Destinations written for the instruction
 ✓ Next instruction can enter E1 during cycle
 Xr Next instruction cannot enter E1 during cycle—read/decode constraint
 Xw Next instruction cannot enter E1 during cycle—write constraint
 Xu Next instruction cannot enter E1 during cycle—other resource conflict

Page: **Change or Add:**

7–29 Change the Instruction Execution in Cycles 2–4 for Load and Store in Table 7–11:

Table 7–11. MPYDP .M-Unit Instruction Constraints

Instruction Execution											
Cycle	1	2	3	4	5	6	7	8	9	10	11
MPYDP	R	R	R	R					W	W	
Instruction Type	Subsequent Same-Unit Instruction Executable										
16 × 16 multiply		Xr	Xr	Xr	✓	✓	✓	Xw	Xw	✓	✓
4-cycle		Xr	Xr	Xr	Xu	Xw	Xw	✓	✓	✓	✓
MPYI		Xr	Xr	Xr	Xu	Xu	Xu	✓	✓	✓	✓
MPYID		Xr	Xr	Xr	Xu	Xu	Xu	✓	✓	✓	✓
MPYDP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
Instruction Type	Same Side, Different Unit, Both Using Cross Path Executable										
Single-cycle		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
Load		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Store		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DP compare		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
2-cycle DP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
Branch		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
4-cycle		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
INTDP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓
ADDDP/SUBDP		Xr	Xr	Xr	✓	✓	✓	✓	✓	✓	✓


Legend:  E1 phase of the single-cycle instruction
 R Sources read for the instruction
 W Destinations written for the instruction
 ✓ Next instruction can enter E1 during cycle
 Xr Next instruction cannot enter E1 during cycle—read/decode constraint
 Xw Next instruction cannot enter E1 during cycle—write constraint
 Xu Next instruction cannot enter E1 during cycle—other resource conflict

Page: **Change or Add:**

7-33 Change the Instruction Execution in Cycle 2 for Load and Store in Table 7-15:

Table 7-15. ADDDP/SUBDP .L-Unit Instruction Constraints

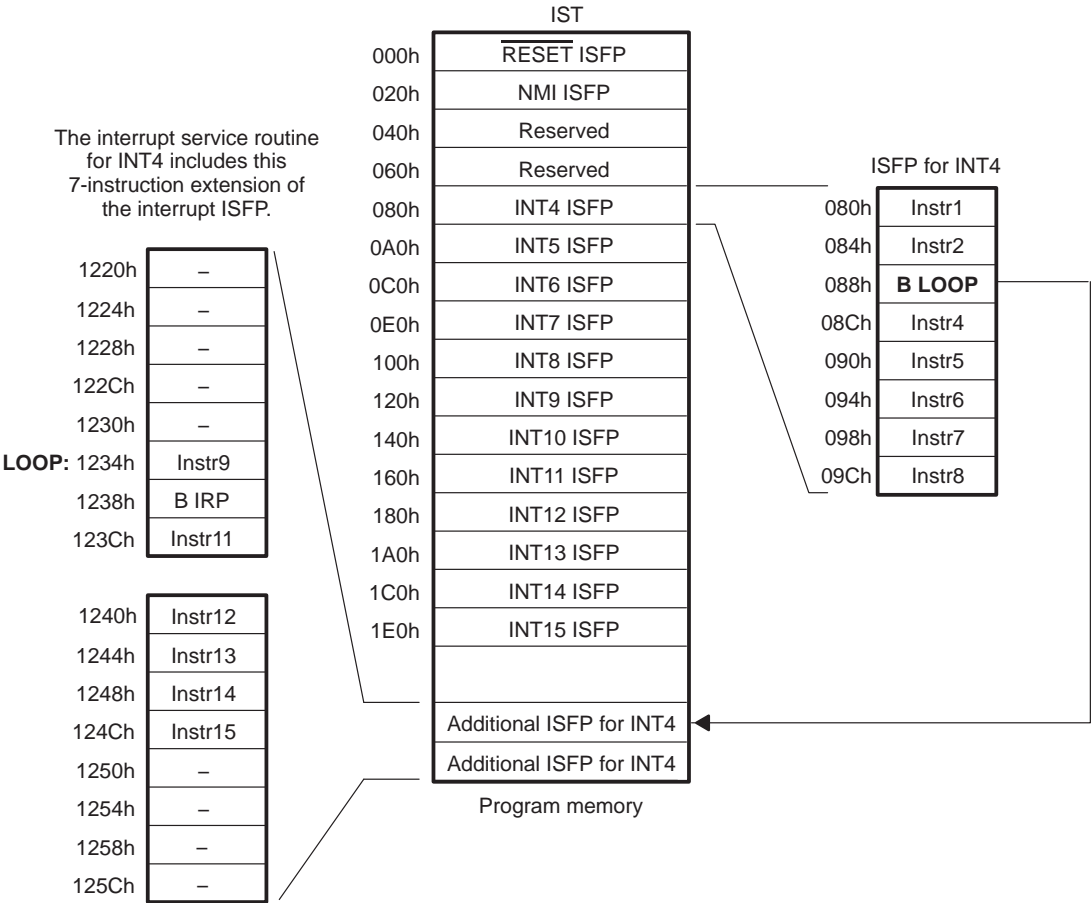
Instruction Execution								
Cycle	1	2	3	4	5	6	7	8
ADDDP/SUBDP	R	R				W	W	
Instruction Type	Subsequent Same-Unit Instruction Executable							
Single-cycle		Xr	✓	✓	✓	Xw	Xw	✓
4-cycle		Xr	Xw	Xw	✓	✓	✓	✓
INTDP		Xrw	Xw	Xw	✓	✓	✓	✓
ADDDP/SUBDP		Xr	✓	✓	✓	✓	✓	✓
Instruction Type	Same Side, Different Unit, Both Using Cross Path Executable							
Single-cycle		Xr	✓	✓	✓	✓	✓	✓
DP compare		Xr	✓	✓	✓	✓	✓	✓
2-cycle DP		Xr	✓	✓	✓	✓	✓	✓
4-cycle		Xr	✓	✓	✓	✓	✓	✓
Load		✓	✓	✓	✓	✓	✓	✓
Store		✓	✓	✓	✓	✓	✓	✓
Branch		Xr	✓	✓	✓	✓	✓	✓
16 × 16 multiply		Xr	✓	✓	✓	✓	✓	✓
MPYI		Xr	✓	✓	✓	✓	✓	✓
MPYID		Xr	✓	✓	✓	✓	✓	✓
MPYDP		Xr	✓	✓	✓	✓	✓	✓

Legend:  E1 phase of the single-cycle instruction
 R Sources read for the instruction
 W Destinations written for the instruction
 ✓ Next instruction can enter E1 during cycle
 Xr Next instruction cannot enter E1 during cycle-read/decode constraint
 Xw Next instruction cannot enter E1 during cycle-write constraint
 Xrw Next instruction cannot enter E1 during cycle-read/decode/write constraint

Page: Change or Add:

8-7 Change Figure 8-3 and the Note that follows:

Figure 8-3. IST With Branch to Additional Interrupt Service Code Located Outside the IST



Note:

The instruction **B LOOP** branches into the middle of a fetch packet and processes code starting at address 1234h. The CPU ignores code from address 1220h-1230h, even if it is in parallel to code at address 1234h.

Page: Change or Add:**8–12** Change the paragraphs in section 8.2:

Bit 1 of CSR is the PGIE bit and holds the previous value of GIE when a maskable interrupt is processed. During maskable interrupt processing, the value of the GIE bit is copied to the PGIE bit, and the GIE bit is cleared. The previous value of the PGIE bit is lost. The GIE bit is cleared during a maskable interrupt to prevent another maskable interrupt from occurring before the device state has been saved. Upon returning from an interrupt, by way of the **B IRP** instruction, the content of the PGIE bit is copied back to the GIE bit. The PGIE bit remains unchanged.

The purpose of the PGIE bit is to record the value of the GIE bit at the time the interrupt processing begins. This is necessary because interrupts are detected in parallel with instruction execution. Typically, the GIE bit is 1 when an interrupt is taken. However, if an interrupt is detected in parallel with an **MVC** instruction that clears the GIE bit, the GIE bit may be cleared by the **MVC** instruction after the interrupt processing begins. Because the PGIE bit records the state of the GIE bit after all instructions have completed execution, the PGIE bit captures the fact that the GIE bit was cleared as the interrupt was taken.

For example, suppose the GIE bit is set to 1 as the sequence of code shown in Example 8–2 is entered. An interrupt occurs, and the CPU detects it just as the CPU is executing the **MVC** instruction that writes a 0 to the GIE bit. Interrupt processing begins. Meanwhile, the 0 is written to the GIE bit as the **MVC** instruction completes. During the interrupt dispatch, this updated value of the GIE bit is copied to the PGIE bit, leaving the PGIE bit cleared to 0. Later, upon returning from the interrupt (using the **B IRP** instruction), the PGIE bit is copied to the GIE bit. As a result, the code following the **MVC** instruction recognizes the GIE bit is cleared to 0, as directed by the **MVC** instruction, despite having taken the interrupt.

Page: **Change or Add:**

8–26 Change section 8.6.2. Add Examples 8–12 and 8–13. The subsequent examples are renumbered accordingly:

8.6.2 Nested Interrupts

Generally, when the CPU enters an interrupt service routine, interrupts are disabled. However, when the interrupt service routine is for one of the maskable interrupts (INT4–INT15), an NMI can interrupt processing of the maskable interrupt. In other words, an NMI can interrupt a maskable interrupt, but neither an NMI nor a maskable interrupt can interrupt an NMI.

There may be times when you want to allow an interrupt service routine to be interrupted by another (particularly higher priority) interrupt. Even though the processor by default does not allow interrupt service routines to be interrupted unless the source is an NMI, it is possible to nest interrupts under software control. **To allow nested interrupts, the interrupt service routine must perform the following initial steps in addition to its normal work of saving any registers (including control registers) that it modifies:**

- 1) The contents of IRP (or NRP) must be saved,
- 2) The contents of PGIE must be saved, and
- 3) The GIE bit must be set to 1.

Prior to returning from the interrupt service routine, the code must restore the registers saved above as follows:

- 1) The GIE bit must be first cleared to 0,
- 2) The PGIE bit's saved value must be restored, and
- 3) The IRP (or NRP) register's saved value must be restored.

Although steps 2 and 3 above may be performed in either order, it is important that GIE is cleared first. This means that GIE and PGIE must be restored with separate writes to CSR. If these bits are not restored separately, then it is possible that PGIE will be overwritten by nested interrupt processing just as interrupts are being disabled.

Example 8–12 illustrates a simple assembly interrupt handler that allows nested interrupts. This example saves its context to the system stack, pointed to by B15. This assumes that the C runtime conventions are being followed. The example code is not optimized, to aid in readability.

Example 8–13 illustrates a C-based interrupt handler that allows nested interrupts. The steps are similar, although the compiler takes care of allocating the stack and saving CPU registers. For more information on using C to access control registers and write interrupt handlers, see the *TMS320C6000 Optimizing C Compiler Users Guide*, SPRU187.

Page: **Change or Add:**

Example 8–12. Assembly Interrupt Service Routine That Allows Nested Interrupts

```

_isr:
    STW    B0, *B15--[4]    ; Save B0, allocate 4 words of stack
    STW    B1, *B15[1]     ; Save B1 on stack

    MVC    IRP, B0
    STW    B0, *B15[2]     ; Save IRP on stack

    MVC    CSR, B0
    STW    B0, *B15[3]     ; Save CSR (and thus PGIE) on stack

    OR     B0, 1, B1
    MVC    B1, CSR         ; Enable interrupts

    ; Interrupt service code goes here.
    ; Interrupts may occur while this code executes.

    MVC    CSR, B0         ;\
    AND    B0, -2, B1      ; |-- Disable interrupts.
    MVC    B1, CSR         ;/   (Set GIE to 0)

    LDW    *B15[3], B0     ; get saved value of CSR into B0
    NOP    4               ; wait for LDW *B15[3] to finish
    MVC    B0, CSR         ; Restore PGIE

    LDW    *B15[2], B0     ; get saved value of IRP into B1
    NOP    4
    MVC    B0, IRP         ; Restore IRP

    B      IRP             ; Return from interrupt
|| LDW    *B15[1], B1      ; Restore B1
    LDW    *++B15[4], B0   ; Restore B0, release stack.
    NOP    4               ; wait for B IRP and LDW to complete.

```

Example 8–13. C Interrupt Service Routine That Allows Nested Interrupts

```

/* c6x.h contains declarations of the C6x control registers */
#include <c6x.h>

interrupt void isr(void)
{
    unsigned old_csr;
    unsigned old_irp;

    old_irp = IRP    ;/* Save IRP                */
    old_csr = CSR    ;/* Save CSR (and thus PGIE) */

    CSR = old_csr | 1 ;/* Enable interrupts      */

    /* Interrupt service code goes here.          */
    /* Interrupts may occur while this code executes */

    CSR = CSR & -2    ;/* Disable interrupts      */
    CSR = old_csr     ;/* Restore CSR (and thus PGIE) */
    IRP = old_irp     ;/* Restore IRP          */
}

```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Copyright © 2004, Texas Instruments Incorporated