Overflow Conditions in CMACD Instruction Execution on NS32GX320

National Semiconductor
Application Note 720
Yair Hadas
August 1990

Overflow Conditions in CMACD Instruction Execution on NS32GX320

AN-720

1.0 INTRODUCTION

This application note describes how to handle overflow occurrence in the CMACD (Complex Multiply and Accumulate Double) instruction in the NS32GX320. It includes a description of cases in which overflow occurs in the CMACD instruction, and provides examples of these cases.

2.0 DESCRIPTION

2.1 A General Description of Overflow in Addition Operations

Overflows occur in addition operations when the carry bit into the sign bit position disagrees with the carry bit out of the sign bit position. When this happens, the correct result is too large to be to represented as a signed integer number, and is often represented by the wrong sign. Thus, the overflow of two positive numbers yields a negative number, while the overflow of two negative numbers yields a positive number.

2.2 Examples of Overflow in Addition Operations

This section provides two examples of overflows in addition operations with two 8-bit numbers. The first example shows what happens when two positive numbers are added. The second example shows what happens when two negative (2's complement) numbers are added.

Example A: Adding Two Positive Numbers

```
Operand 1:  01000101
Operand 2:  01001010
Result:     10001111
            | |
            |→ carry bit = 1
            |
            → carry bit = 0  →  Overflow
```

This example shows how adding two positive operands (01000101 = 69 decimal, 01001010 = 74 decimal) produces a negative result (10001111 = −113 decimal).

Example B: Adding Two Negative Numbers

```
Operand 1:  10001101
Operand 2:  10000111
Result:     00010100
            | |
            |→ carry bit = 0
            |
            → carry bit = 1  →  Overflow
```

This example shows how adding two negative operands (10001101 = −115 decimal, 10000111 = −121 decimal) produces a positive result: (00010100 = 20 decimal).

3.0 STANDARD OVERFLOW IN CMACD

The Complex Multiply and Accumulate Double (CMACD) instruction reads two double-word source operands, representing complex numbers. Each operand consists of a signed 16-bit real part in the low-order word and a signed 16-bit imaginary part in the high-order word. The result consists of a signed 32-bit real part in R0, and a signed 32-bit imaginary part in R1.

The instruction syntax is:

```
CMACD  src1,   src2
       gen     gen
       read.D  read.D
```

Assuming:

src1 =	A2 (imagin)	A1 (real)
	31 16	15 0
src2 =	B2 (imagin)	B1 (real)
	31 16	15 0

The CMACD result is:

R0 := $\boxed{R0 + A1 * B1 - A2 * B2}$ -- Real result

R1 := $\boxed{R1 + A1 * B2 + A2 * B1}$ -- Imaginary result

The CMACD instruction consists of three ADD operations and one SUB operation. The overflow can occur during each of these operations.

Following is the order of additions and subtractions in the execution of a CMACD instruction:

a. R1 + A2 * B1

b. R0 − A2 * B2

c. (R1 + A2 * B1) + A1 * B2

d. (R0 − A2 * B2) + A1 * B1

The following section gives 8 examples of this standard type of overflow occurrence. Section 4.0 will give 4 examples of a more complicated type of overflow occurrence.

TL/EE11064

RRD-B30M75/Printed in U. S. A.

4.0 EXAMPLES OF STANDARD OVERFLOW

Example 1: Overflow on R1 + A2 * B1 when A2 * B1 is Positive

In this operation, an overflow occurs when the product of A2 and B1 is added to R1.

```
#    r1           = 60000000
#    a2 * b1      = 31000000
#    r1 + a2 * b1 = 91000000  →  overflow
ovf1:  movd   $h′ 60000000, r1
       movd   $h′ 70000001, r2
       movd   $h′ 00017000, r3
       cmacd  r2, r3
```

Here is what happens when the operation reaches the CMACD. (Note that A2 and B1 appear in italics.):

```
src1         = 0111000000000000000000 0000000001
src2         = 00000000000000010111000000000000
A2 * B1      = 00110001000000000000000000000000
r1           = 01100000000000000000000000000000
r1 + A2 * B1 = 10010001000000000000000000000000
               | |
               |→ carry bit = 1
               |
                → carry bit = 0  →  Overflow
```

Example 2: Overflow on R1 + A2 * B1 when A2 * B1 is Negative

In this operation, an overflow occurs when the product of A2 and B1 is added to R1.

```
#    r1           = b0000000
#    a2 * b1      = c000ffff
#    r1 + a2 * b1 = 7000ffff  →  overflow
ovf2:  movd   $h′ b0000000, r1
       movd   $h′ 7fff0001, r2
       movd   $h′ 00018001, r3
       cmacd  r2, r3
```

Here’s what happens when the operation reaches the CMACD. (Note that A2 and B1 appear in italics.):

```
src1         = 01111111111111110000000000000001
src2         = 00000000000000011000000000000001
a2 * b1      = 11000000000000001111111111111111
r1           = 10110000000000000000000000000000
r1 + a2 * b1 = 01110000000000001111111111111111
               | |
               |→ carry bit = 0
               |
                → carry bit = 1  →  Overflow
```

Example 3: Overflow on (R1 + A2 * B1) +A1 * B2 when A1 * B2 is Positive

In this operation an overflow occurs when the product of A1 and B2 is added to R1 + A2 * B1.

```
#    r1                     = 20000000
#    a2 * b1                = 31000000
#    r1 * a2 * b1           = 51000000
#    a1 * b2                = 40000000
#    r1 + a2 * b1 + a1 * b2 = 91000000  →  overflow
ovf3:  movd   $h′ 20000000, r1
       movd   $h′ 70008000, r2
       movd   $h′ 80007000, r3
       cmacd  r2, r3
```

Example 4: Overflow on (R1 + A2 * B1) + A1 * B2 when A1 * B2 is Negative

In this operation an overflow occurs when the product of A1 and B2 is added to R1 + A2 * B1.

```
#    r1                    = c0000000
#    a2 * b1               = c000ffff
#    r1 * a2 * b1          = 8000ffff
#    a1 * b2               = f0000000
#    r1 + a2 * b1 + a1 * b2 = 7000ffff  →  overflow
ovf4:  movd   $h′ c0000000, r1
       movd   $h′ 7fff4000, r2
       movd   $h′ c0008001, r3
       cmacd  r2, r3
```

Example 5: Overflow on R0 − A2 * B2 when A2 * B2 is Positive

In this operation an overflow occurs when the product of A2 and B2 is subtracted from R0.

```
#    r0          = 60000000
#    a2 * b2     = c000ffff
#    r0 - a2 * b2 = 9fff0001  →  overflow
ovf5:  movd   $h′ 60000000, r0
       movd   $h′ 7fff0001, r2
       movd   $h′ 80010001, r3
       cmacd  r2, r3
```

Example 6: Overflow on R0 − A2 * B2 when A2 * B2 is positive.

In this operation an overflow occurs when the product of A2 and B2 is subtracted from R0.

```
#    r0          = b0000000
#    a2 * b2     = 3fff0001
#    r0 - a2 * b2 = 7000ffff  →  overflow
ovf6:  movd   $h′ b0000000, r0
       movd   $h′ 7fff0001, r2
       movd   $h′ 7fff0001, r3
       cmacd  r2, r3
```

Example 7: Overflow on (R0 − A2 * B2) + A1 * B1 when A1 * B1 is positive.

In this operation an overflow occurs when the product of A1 and B1 is added to R0 − A2 * B2.

```
#    r1                    = 70000000
#    a2 * b2               = 10000000
#    r0 - a2 * b2          = 60000000
#    a1 * b1               = 31000000
#    r0 - a2 * b2 + a1 * b1 = 91000000  →  overflow
ovf7:  movd   $h′ 70000000, r0
       movd   $h′ 40007000, r2
       movd   $h′ 40007000, r3
       cmacd  r2, r3
```

Example 8: Overflow on (R0 − A2 * B2) + A1 * B1 when A1 * B1 is negative.

In this operation an overflow occurs when the product of A1 and B1 is added to R0 − A2 * B2.

```
#    r0                    = c0000000
#    a2 * b2               = 10000000
#    r0 - a2 * b2          = b0000000
#    a1 * b1               = c000ffff
#    r0 - a2 * b2 + a1 * b1 = 7000ffff  →  overflow
ovf8:  movd   $h′ c0000000, r0
       movd   $h′ 40007fff, r2
       movd   $h′ 40008001, r3
       cmacd  r2, r3
```

5.0 COMPLICATED OVERFLOW IN CMACD

There are two cases when the final result is correct although the overflow flag is set. These are the cases in which an overflow occurs in one of the first two operations, but the third or fourth operation compensates for the overflow. In these cases the overflow depends on the order of the operands. There will be no overflow if the source operands are exchanged. These two cases of complicated overflow are:

- R1 + A2 * B1 causes an overflow.

  (R1 + A2 * B1) + A1 * B2 compensates for the overflow.
- R0 − A2 * B2 causes an overflow.

  (R0 − A2 * B2) + A1 * B1 compensates for the overflow.

The following section presents 4 examples of these cases:

6.0 EXAMPLES OF COMPLICATED OVERFLOW

Example 1: Intermediate overflow on R1 + A2 * B1 when R1 + A2 * B1 + A1 * B2 does not cause an overflow. This assumes that A2 * B1 is positive and A1 * B2 is negative. If operands A and B were exchanged, there would not have been an overflow.

```
#    r1                      = 60000000
#    a2 * b1                 = 31000000
#    r1 + a2 * b1            = 91000000 →  overflow
#    a1 * b2                 = c000ffff
#    r1 + a2 * b1 + a1 * b2 = 5100ffff →  no overflow

ovf9:  movd   $h' 60000000, r1
       movd   $h' 70007fff, r2
       movd   $h' 80017000, r3
       cmacd  r2, r3
```

Here's what happens when the operation reaches the CMACD. Note that A1 and B1 are in italics, while A2 and B2 are not.

```
src1         = 01110000000000000111111111111111
src2         = 10000000000000010111000000000000
a2 * b1      = 00110001000000000000000000000000
r1           = 01100000000000000000000000000000
r1 + a2 * b1 = 10010001000000000000000000000000
               | |
               |→ carry bit = 1
               |
               → carry bit = 0 →  Overflow
a1 * b2      = 11000000000000001111111111111111
final result = r1 + a2 * b1 + a1 * b2
final result = 01010001000000001111111111111111
```

The final result is correct although there was an intermediate overflow.

Now we check the same instruction when we exchange the order of the operands. Note that exchanging the operands prevents overflow (A1 and B1 are still in italics).

```
src1         = 10000000000000010111000000000000
src2         = 01110000000000000111111111111111
a2 * b1      = 110000000000000011111111111111111
r1           = 01100000000000000000000000000000
r1 + a2 * b1 = 00100000000000001111111111111111
               | |
               |→ carry bit = 1
               |
               → carry bit = 1 →  No Overflow
a1 * b2      = 00110001000000000000000000000000
final result = r1 + a2 * b1 + a1 * b2
final result = 01010001000000001111111111111111
```

We can see that the result is similar to the previous and no overflow occurred during the instruction operation.

Example 2: Intermediate overflow on R1 + A2 * B1 when R1 + A2 * B1 + A1 * B2 do not cause an overflow. This assumes that A2 * B1 is negative and A1 * B2 is positive. If operands A and B were exchanged there wouldn't have been an overflow.

```
 #    r1                      = b0000000
 #    a2 * b1                 = c000ffff
 #    r1 + a2 * b1            = 7000ffff  →  overflow
 #    a1 * b2                 = 10000000
 #    r1 + a2 * b1 + a1 * b2 = 8000ffff  →  no overflow

 ovfl0:  movd   $h′ b0000000, r1
         movd   $h′ 7fff4000, r2
         movd   $h′ 40008001, r3
         cmacd  r2, r3
```

Example 3: Intermediate overflow on R0 − A2 * B2 when R0 − A2 * B2 + A1 * B1 do not overflow. This assumes that A2 * B2 is negative and A1 * B1 is negative. If operands A and B were exchanged there wouldn't have been an overflow.

```
 #    r0                      = 60000000
 #    a2 * b2                 = c000ffff
 #    r0 - a2 * b2            = 9fff0001  →  overflow
 #    a1 * b1                 = c000ffff
 #    r0 - a2 * b2 + a1 * b1 = 60000000  →  no overflow

 ovfl1:  movd   $h′ 60000000, r0
         movd   $h′ 7fff8001, r2
         movd   $h′ 80017fff, r3
         cmacd  r2, r3
```

Example 4: Intermediate overflow on R0 − A2 * B2 when R0 − A2 * B2 + A1 * B1 do not overflow. This assumes that A2 * B2 is positive and A1 * B1 is positive. If operands A and B were exchanged there wouldn't have been an overflow.

```
 #    r0                      = b0000000
 #    a2 * b2                 = 3fff0001
 #    r0 - a2 * b2            = 7000ffff  →  overflow
 #    a1 * b1                 = 10000000
 #    r0 - a2 * b2 + a1 * b1 = 8000ffff  →  no overflow

 ovfl2:  movd   $h′ b0000000, r0
         movd   $h′ 7fffc000, r2
         movd   $h′ 7fffc000, r3
         cmacd  r2, r3
```

AN-720

Overflow Conditions in CMACD Instruction Execution on NS32GX320

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor
Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor
GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor
Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor
Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores
Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor
(Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.