Chapter 13
Numerical Issues

Learning Objectives

- Numerical issues and data formats.
- Fixed point.
- Fractional number.
- Floating point.
- Comparison of formats and dynamic ranges.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Numerical Issues and Data Formats

C6000 Numerical Representation

Fixed point arithmetic:

- 16-bit (integer or fractional).
- Signed or unsigned.

Floating point arithmetic:

- 32-bit single precision.
- 64-bit single precision.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- For simplicity a 4-bit representation is used:

Binary Number	2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	0	0	
Unsigned integer numbers	0	0	0	0	0

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- **For simplicity a 4-bit representation is used:**

Binary Number	2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	0	1	
Unsigned integer numbers	0	0	0	0	0
	0	0	0	1	1

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- For simplicity a 4-bit representation is used:

Binary Number	2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	1	0	
Unsigned integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- For simplicity a 4-bit representation is used:

Binary Number	2^3	2^2	2^1	2^0	Decimal Equivalent
	1	1	1	1	
Unsigned integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	8
	1	0	0	1	9
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- For simplicity a 4-bit representation is used:

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	0	0	
Signed integer numbers	0	0	0	0	0

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- **For simplicity a 4-bit representation is used:**

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	0	1	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- For simplicity a 4-bit representation is used:

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	0	0	1	0	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- **For simplicity a 4-bit representation is used:**

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	0	1	1	1	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- For simplicity a 4-bit representation is used:

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	1	0	0	0	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	-8

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- For simplicity a 4-bit representation is used:

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	1	0	0	1	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	-8
	1	0	0	1	-7

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Definition

- For simplicity a 4-bit representation is used:

Binary Number	-2^3	2^2	2^1	2^0	Decimal Equivalent
	1	1	1	1	
Signed integer numbers	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	-8
	1	0	0	1	-7
	1	0	1	0	-6
	1	0	1	1	-5
	1	1	0	0	-4
	1	1	0	1	-3
	1	1	1	0	-2
	1	1	1	1	-1

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Problems

- **The following equation is the basis of many DSP algorithms (See Chapter 1):**

$$y(n)=\sum_{k=0}^{N-1} a(k)x(n-k)$$

- **Two problems arise when using signed and unsigned integers:**
  - **Multiplication overflow.**
  - **Addition overflow.**

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Multiplication Overflow

- 16-bit x 16-bit = 32-bit
- Example: using 4-bit representation

				0	0	1	1
			x	1	0	0	0
0	0	0	1	1	0	0	0

$$\begin{array}{r} 3 \\ \times \quad 8 \\ \hline 24 \end{array}$$

- 24 cannot be represented with 4-bits.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Addition Overflow

- 32-bit + 32-bit = 33-bit
- Example: using 4-bit representation

		1	0	0	0
	+	1	0	0	0
	1	0	0	0	0

	8
+	8
	16

- 16 cannot be represented with 4-bits.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fixed Point Arithmetic - Solution

- **The solutions for reducing the overflow problem are:**
  - **Saturate the result.**
  - **Use double precision result.**
  - **Use fractional arithmetic.**
  - **Use floating point arithmetic.**

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Solution - Saturate the result

- **Unsigned numbers:**
  - If A x B ≤ 15 → result = A x B
  - If A x B > 15 → result = 15

				0	0	1	1	3
			x	1	0	0	0	8
0	0	0	1	1	0	0	0	24
Saturated				1	1	1	1	15

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Solution - Saturate the result

- **Signed numbers:**
  - **If $-8 \le A \times B \le 7 \rightarrow$ result = A x B**
  - **If $A \times B > 7 \rightarrow$ result = 7**
  - **If $A \times B < -8 \rightarrow$ result = -8**

				0	0	1	1		3
			x	1	0	0	0		-8
1	1	1	0	1	0	0	0		-24
Saturated				1	0	0	0		-8

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Solution - Double precision result

- For a 4-bit x 4-bit multiplication hold the result in an 8-bit location.
- Problems:
  - Uses more memory for storing data.
  - If the result is used in another multiplication the data needs to be represented into single precision format (e.g. prod = prod x sum).
  - Results need to be scaled down if it is to be sent to an A to D converter.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

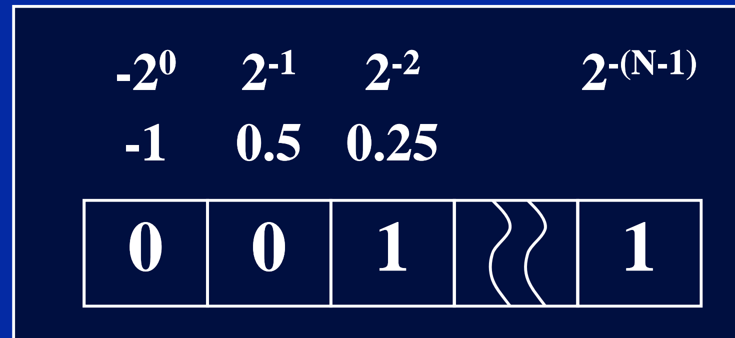
Solution - Fractional arithmetic

- **If A and B are fractional then:**
  - **A x B < min(A, B)**
  - **i.e. The result is less than the operands hence it will never overflow.**
- **Examples:**
  - **0.6 x 0.2 − 0.12 (0.12 < 0.6 and 0.12 < 0.2)**
  - **0.9 x 0.9 = 0.81 (0.81 < 0.9)**
  - **0.1 x 0.1 = 0.01 (0.01 < 0.1)**

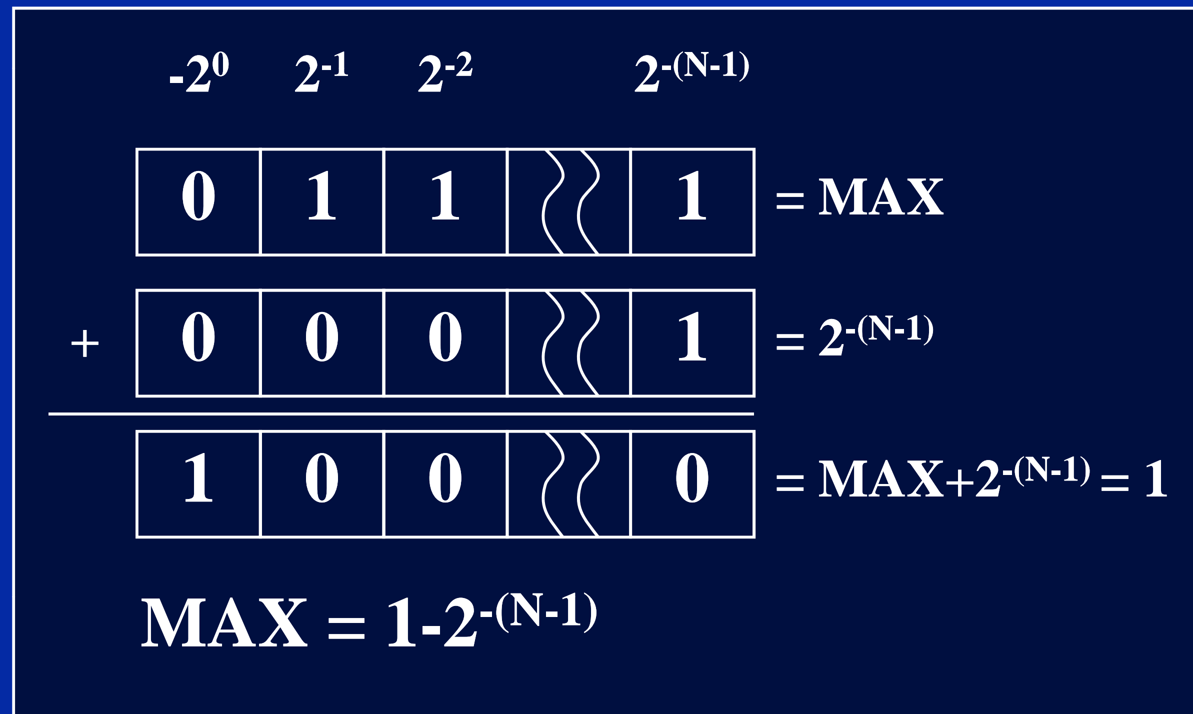
Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fractional numbers

- **Definition:**



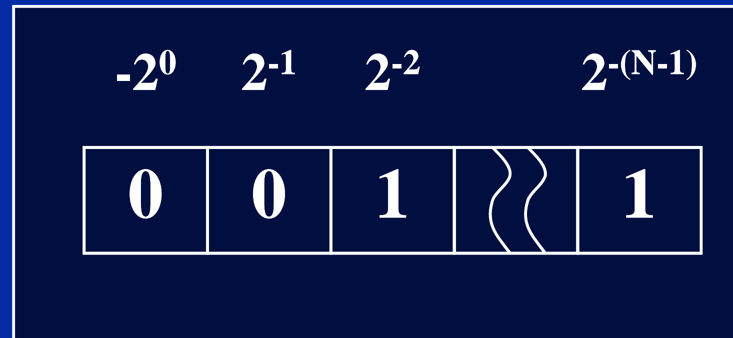
- **What is the largest number?**
- **Largest Number:**



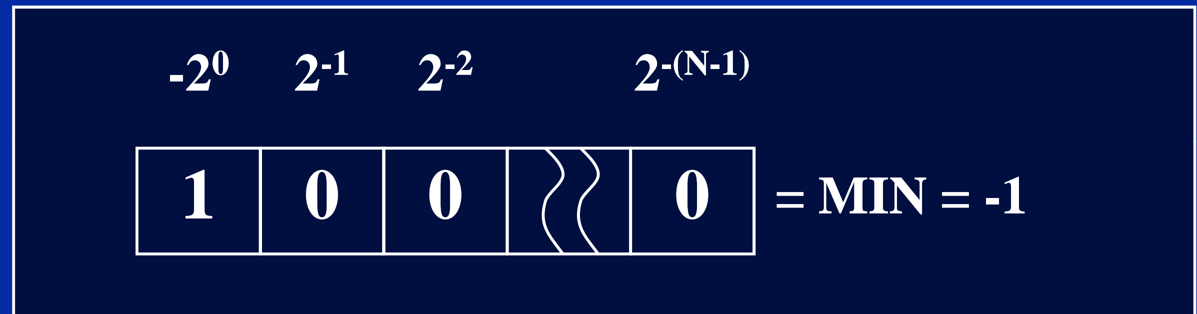
Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fractional numbers

- **Definition:**



- **What is the smallest number?**
- **Smallest Number:**



- **For 16-bit representation:**
  - **MAX = $1 - 2^{-15} = 0.999969$**
  - **MIN = -1**
  - **$-1 \leq x < 1$**

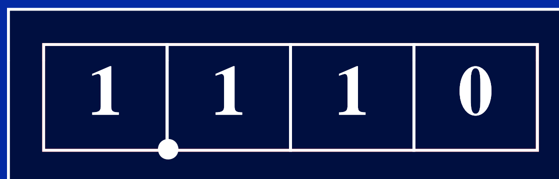
Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fractional numbers - Sign Extension

				a=	0	1	1	0	= 0.5 + 0.25 = 0.75
		x		b=	1	1	1	0	= -1 + 0.5 + 0.25 = -0.25
					0	0	0	0	
				0	1	1	0	.	
			0	1	1	0	.	.	
		1	0	1	0	.	.	.	
	1	1	1	1	0	1	0	0	

Sign extension

- To keep the same resolution as the operands we need to select these 4-bits:



Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fractional numbers - Sign Extension

				a=	0	1	1	0	= 0.5 + 0.25 = 0.75
			x	b=	1	1	1	0	= -1 + 0.5 + 0.25 = -0.25

0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	.
0	0	0	1	1	0	.	.
1	1	0	1	0	.	.	.
1	1	1	1	0	1	0	0

Sign extension bits

Sign extension

- The way to do it is to shift left by one bit and store upper 4-bits or right shift by three and store the lower 4-bits:

1	1	1	0

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

15-bit * 15-bit Multiplication

CPU

Q15 | s. | x | x | x | x | x | x | x | x | x | x | x | x | x | x | x

x Q15 | s. | y | y | y | y | y | y | y | y | y | y | y | y | y | y | y

```
MPY    A3,A4,A6
NOP
```

Q30 | s | s. | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z | z

Store to Data Memory

Q15 | s. | y | y | y | y | y | y | y | y | y | y | y | y | y | y | y

```
SHR    A6,15,A6
STH    A6,*A7
```


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

‘C6000 C Data Types

Type	Size	Representation
char, signed char	8 bits	ASCII
unsigned char	8 bits	ASCII
short	16 bits	2’s complement
unsigned short	16 bits	binary
int, signed int	32 bits	2s complement
unsigned int	32 bits	binary
long, signed long	40 bits	2’s complement
unsigned long	40 bits	binary
enum	32 bits	2’s complement
float	32 bits	IEEE 32-bit
double	64 bits	IEEE 64-bit
long double	64 bits	IEEE 64-bit
pointers	32 bits	binary

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fractional numbers - Sign Extension

- ## Pseudo assembly language:

```
A0 = 0x80000000                        ; initial value
A1 = 0.5                               ; initial value
A2 = 0.5                               ; initial value
A3 = 0                                 ; initial value

    MPY  A1, A2, A3                    ; A3 = 0x10000000
    SHL  A3,1,A3                       ; A3 = 0x20000000
    STH  A3, *A0                       ; 0x2000 -> 0x80000000

or

    MPY  A1, A2, A3                    ; A3 = 0x10000000
    SHR  A3,15,A3                      ; A3 = 0x00002000
    STL  A3, *A0                       ; 0x2000 -> 0x80000000
```

- ## Pseudo ‘C’ language:

```
short     a, b, result;
int       prod;

prod = a * b;
prod = prod >> 15;
result = (short) prod;
```


Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Fractional numbers - Problems

- There are some problems that need to be resolved when using fractional numbers.
- These are:
  - Result of -1 x -1 = 1
  - Accumulative overflow.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Problem of -1 x -1

- **We have seen that:**
  - $-1 \leq x < 1$
  - $-1 \times -1 = 1$ **which cannot be represented.**
- **Solution:**
  - **There are two instructions that saturate the result if you have -1 x -1:**

SMPY	SMPYH

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Problem of -1 x -1

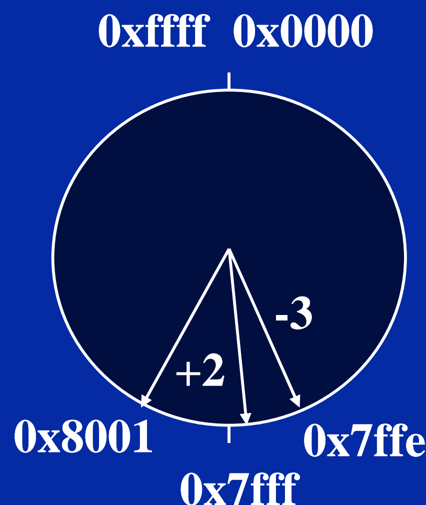
- **In one cycle these instructions do the following:**
  - **Multiply.**
  - **Shift left by 1-bit.**
  - **Saturate if the sign bits are 01.**
- **It can be shown that:**

	Result of MPY(H)	Result of SMPY(H)
Positive Result	00.xxx-xb	0.xxx-xb
Negative Result	11.xxx-xb	1.xxx-xb
-1 x -1 Result	01.xxx-xb	0.xxx-xb

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Problem of Accumulative Overflow

- **Result:** $$y(n)=\sum_{k=0}^{99} a(k)x(n-k)$$ = 0x7ffe



- **But intermediate values overflow:**

y_{97} = 0x7fff $\quad$ y_{98} = 0x8001 $\quad$ y_{99} = 0x7ffe

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Problem of Accumulative Overflow

- **Solutions:**
  - (1) Saturate the intermediate results by using these add instructions:

    SADD **SSUB**

    If saturation occurs the SAT bit in the CSR is set to 1. You must clear it.

  - (2) Use guard bits:

    e.g. ADD A1, A2, A1:A0

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Problem of Accumulative Overflow

- **Solutions:**

  (3) Do nothing if the system is <u>Non-Gain:</u>

  With a non-gain system the final result is always less than unity.

  Example system:

$$y(n) = \sum_{k=0}^{99} a(k)x(n-k)$$

  This will be non-gain if:

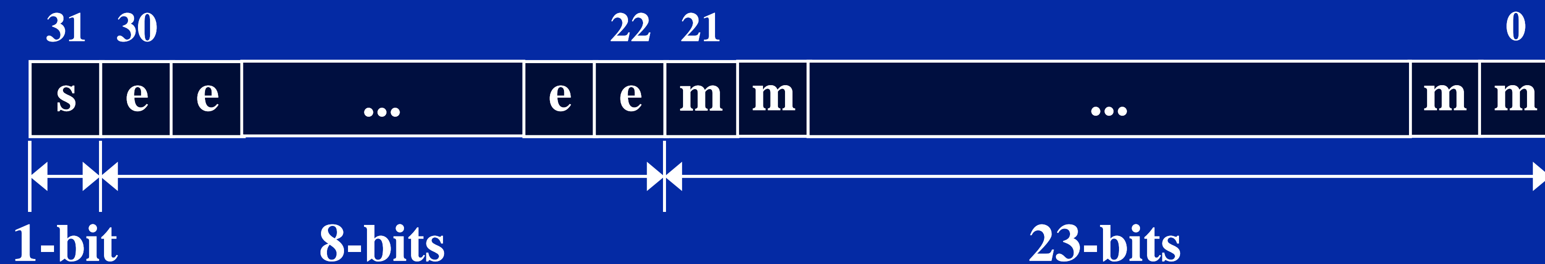
$$\sum_{k=0}^{99} |a(k)| < 1$$

$$|x(i)| \leq 1$$

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating Point Arithmetic

- The C67xx support both single and double precision floating point formats.
- The single precision format is as follows:



s = sign bit

e = exponent (8-bit biased : -127)

m = mantissa (23-bit normalised fraction)

$$value = (-1)^{sign} * (1.mantissa) * 2^{(exponent-127)}$$

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating Point Arithmetic Example

- **Example: Conversion between integer and floating point.**

```
int dd = 0x6000 0000;
```

- **Convert ‘dd’ to the IEEE floating point format:**

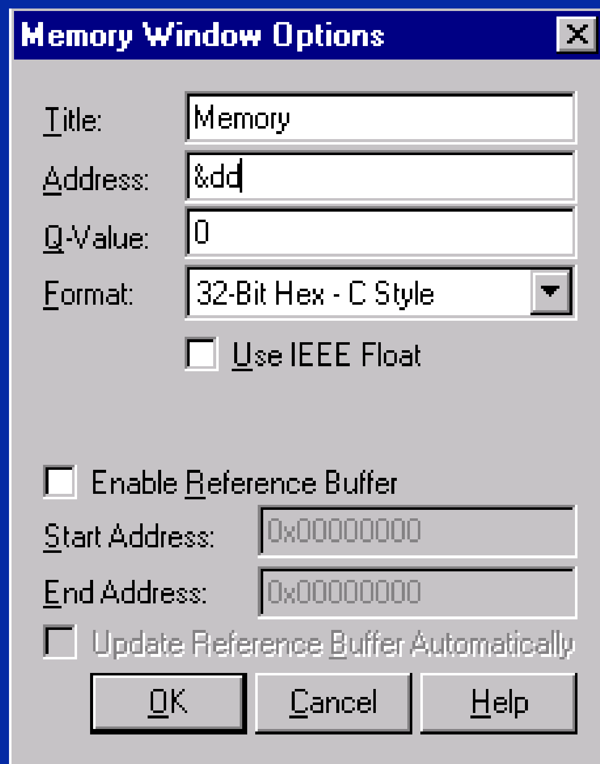
```
flot1 = (float) dd;
```

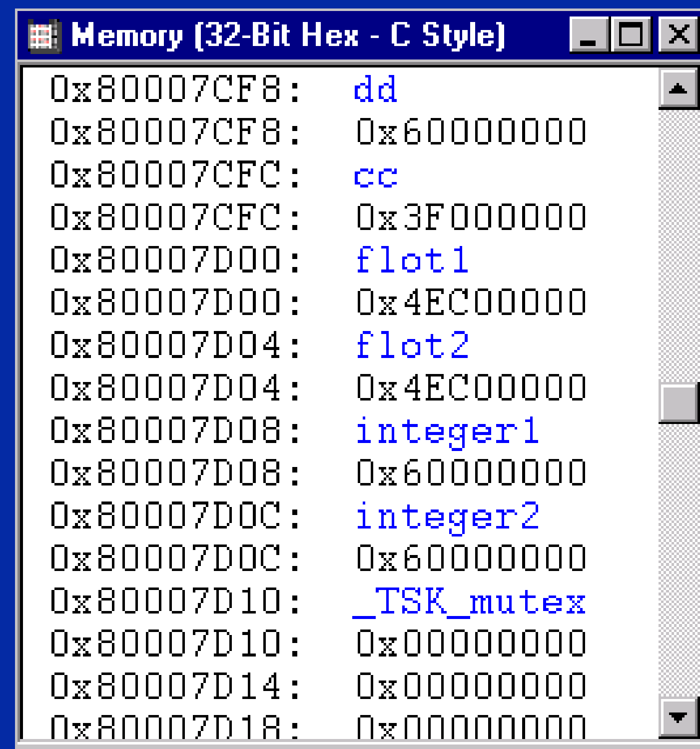

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating Point Arithmetic Example

To view the value of "flot1" use:

View: Memory:Address= &flot1





We find: **flot1 = 0x4EC0 0000**

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating Point Arithmetic Example

- **Let us check to see if we have the same number:**

4				E				C				0				0				0				0				0			
0	1	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s	exponent								mantissa																						

s = 0

e = 10011101b = 128+16+8+4+1 = 157

m = 0.100b = 0.5

float1 $= (-1)^0 * (1.5) * 2^{(157-127)} = 1.5 * 2^{30}$

$= 1610612736$ **decimal**

= 0x6000 0000

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating Point Arithmetic Example

- The previous example can be seen in:
  - numerical.pjt
  - numerical.ws
- Use the mixed mode display to see the assembly code.

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating Point IEEE Standard

- **Special values:**

s	e	m	Number
0	0	0	0
1	0	0	-0
s	0	≠0	$(-1)^s * 0.m * 2^{-126}$
s	0<e<255	m	$(-1)^s * 1.m * 2^{e-127}$
0	255	0	+∞
1	255	0	-∞
s	255	≠0	NaN (not a number)

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating Point IEEE Standard

$$value = (-1)^{sign} * (1.mantissa) * 2^{(exponent-127)}$$

- **Dynamic range:**
  - **Largest positive number:**
    - **e(max) = 255,**
    - **m(max) = $1-2^{-(23-1)}$**
    - **max = $[1 + (1 - 2^{-24})] * 2^{255-127}$**
      = $3.4 * 10^{38}$
  - **Smallest positive number:**
    - **e(min) = 0,**
    - **m(min) = 0.5 (normalised 0.100…0b)**
    - **min = $1.5 * 2^{-127} = 8.816 * 10^{-39}$**

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating Point IEEE Standard

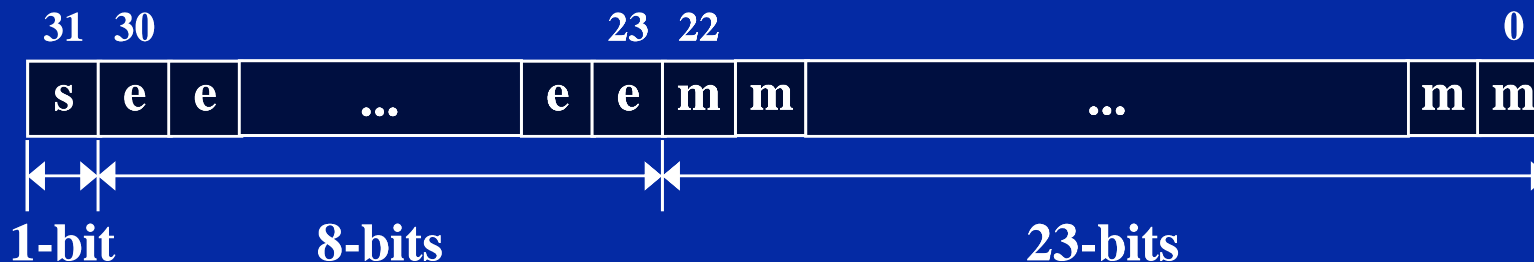
$$value = (-1)^{sign} * (1.mantissa) * 2^{(exponent-127)}$$

- **Dynamic range:**
  - **Largest negative number:**
    - e(max) = 255,
    - m(max) = $1\text{-}2^{-24}$
    - max $= [-1 + (1 - 2^{-24})] * 2^{255-127}$
      $= -3.4 * 10^{38}$
  - **Smallest negative number:**
    - e(min) = 0,
    - m(min) = 0.5 (normalised 1.100…0b)
    - min $= -1.5 * 2^{-127} = -8.816 * 10^{-39}$

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

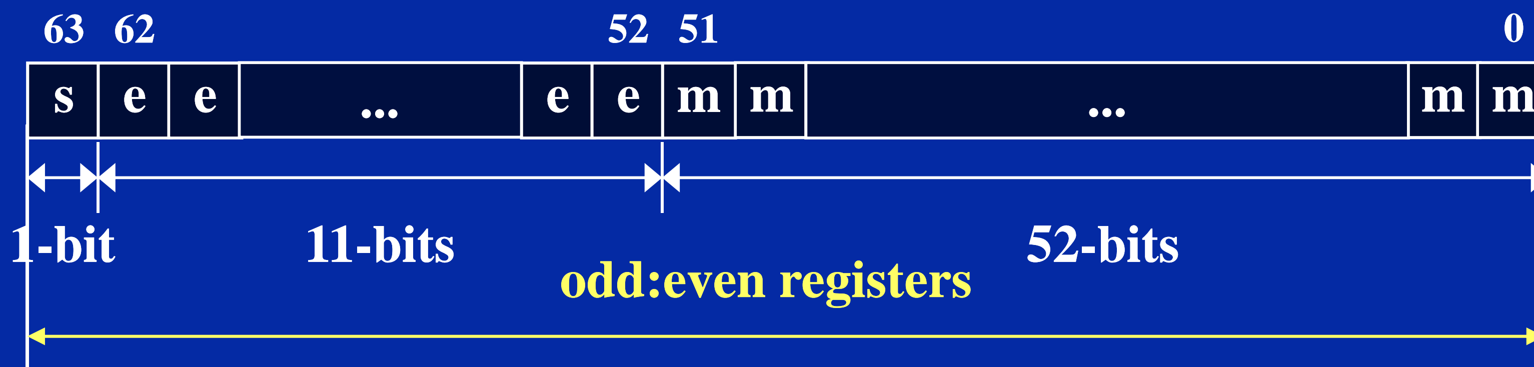
Floating/Fixed Point Summary

- Floating point single precision:



$$value = (-1)^s * 1.m * 2^{e-127}$$

- Floating point double precision:



$$value = (-1)^s * 1.m * 2^{e-1023}$$

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating/Fixed Point Summary (Short: N = 16; Int: N = 32)

- **Unsigned integer:**

2^{N-1}		2^1	2^0
x	...	x	x

- **Signed integer:**

-2^{N-1}		2^1	2^0
x	...	x	x

- **Signed fractional:**

-2^0	2^1	2^2		$2^{-(N-1)}$
x	x	x	...	x

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Floating/Fixed Point Dynamic Range

	Floating Point Single Precision	Fixed Point			
		Integer		Fractional	
		16-bit	32-bit	16-bit	32-bit
Largest Number (positive)	3.4×10^{38}	$2^{16} - 1$	$2^{32} - 1$	$1-2^{-15}$	$1-2^{31}$
Smallest Number (positive)	5.8×10^{-38}	1	1	2^{-15}	2^{-31}
Smallest Number (negative)	-3.4×10^{38}	-2^{16}	-2^{32}	-1	-1

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Numerical Issues - Useful Tips

- **Multiply by 2:** Use shift left
- **Divide by 2:** Use shift right
- **Log_2N:** Use shift
- **Sine, Cosine, Log:** Use look up tables
- **To convert a fractional number to hex:**
  - Num x 2^{15}
  - Then convert to hex

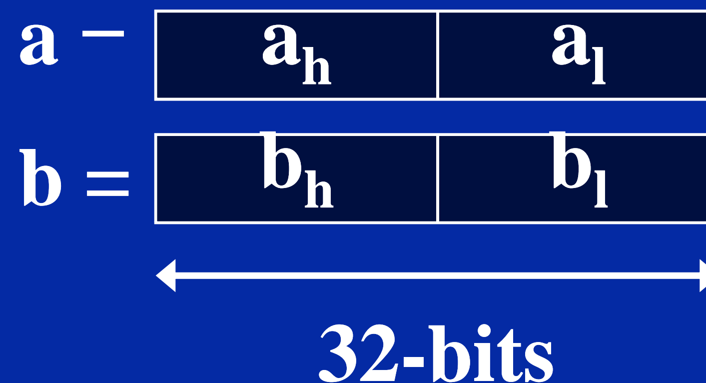
e.g: convert 0.5 to hex

- $0.5 \times 2^{15} = 16384$
- $(16384)_{dec} = (0x4000)_{hex}$

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Numerical Issues - 32-bit Multiplication

- It is possible to perform 32-bit multiplication using 16-bit multipliers.
- Example: c = a x b (with 32-bit values).



$$a * b = (a_h << 16 + a_l) * (b_h << 16 + b_l)$$

$$= [(a_h * b_h) << 32] + [(a_l * b_h) << 16] +$$

$$[(a_h * b_l) << 16] + [a_l * b_l]$$

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Links

- **Further reading:**
  - **Understanding TMS320C62xx DSP Single-precision Floating-Point Functions: \Links\spra515.pdf**
  - **TMS320C6000 Integer Division: \Links\spra707.pdf**

Dr. Naim Dahnoun, Bristol University, (c) Texas Instruments 2002

Chapter 13

Numerical Issues

- End -