

# Lecture 5 Fixed Point vs Floating Point

## Objectives:

- Understand fixed point representations
- Understand scaling, overflow and rounding in fixed point
- Understand Q-format
- Understand TMS320C67xx floating point representations
- Understand relationship between the two in C6x architecture

Reference: "What Every Computer Scientist Should Know About Floating-Point Arithmetic" by David Goldberg ACM Computing Surveys 23, 5 (March 1991).

# Q-Format number representation

- N-bit fixed point, 2's complement number is given by:

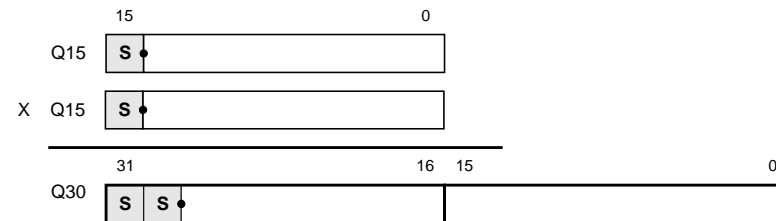
$$x = -b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \dots + b_12^1 + b_02^0$$

- Difficult to work with due to possible overflow & scaling problems
- Often normalise number to some fractional representation (e.g. between ± 1)

$$x' = -b_{N-1}2^0 + b_{N-2}2^{-1} + \dots + b_12^{N-2} + b_02^{N-1}$$

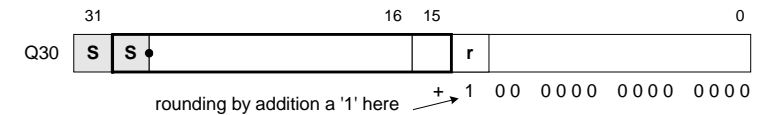
# Q-format notation

- Q-format representation:
  - if N=16, 15 bit fractional representation ⇒ Q15 format
- Rule:
  - $Q_m + Q_m \Rightarrow Q_m$
  - $Q_m \times Q_n \Rightarrow Q_{m+n}$
- Assume 16-bit data format,  $Q15 \times Q15 \Rightarrow Q30$



# How to store Q30 number to 16-bit memory?

- Storing Q30 number to 16-bit memory requires rounding or truncation:



- Rounding:
  - if  $r = 0$ , round down,
  - $r = 1$ , round up

```

MPY  A3,A4,A6 ; A3 x A4 → A6
NOP          ; Delay slot
ADDK 4000h,A6 ; rounding add
SHR  A6,15,A6 ; truncate bottom 15 bits
STH  A6,*A7   ; A6 → mem[A7]
    
```

## Avoid overflow with SADD

- SADD - saturation add instruction
- Always clip to max (or min) possible
- Set bit 9 of the CSR register to indicate saturation has occurred

**Example**      SADD .L1    A1, A2, A3

	Before instruction	1 cycle after instruction	2 cycles after instruction
A1	4367 71F2h	4367 71F2h	4367 71F2h
A2	5A2E 51A3h	5A2E 51A3h	5A2E 51A3h
A3	XXXX XXXXh	7FFF FFFFh	7FFF FFFFh
CSR	0001 0100h	0001 0100h	0001 0300h Saturated

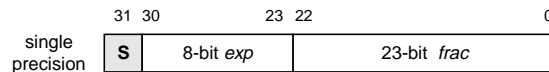
## Safe add routine in C to avoid overflow

```
short safe_add(short A, short B, int *status)
{
    int X, Y, result, SAT_BIT;
    X = A << 16;
    Y = B << 16;
    result = _sadd(X, Y);
    SAT_BIT = GET_REG_BIT(CSR, 9);
    if(SAT_BIT == 1){
        //Overflow Occured
        RESET_REG_BIT(CSR, 9);    //Reset Sat Bit
        *status = 1;
    }
    else
        *status = 0;

    return (result >> 16);
}
```

## Single Precision Floating Point number

- Easy (and lazy) way of dealing with scaling problem
- 32-bit single precision floating point:



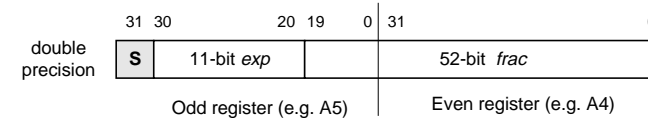
$$x = -1^s \times 2^{exp-127} \times 1.frac$$

$$1.175 \times 10^{-38} < |x| < 1.7 \times 10^{38}$$

- MSB is sign-bit (same as fixed point)
- 8-bit exponent in bias-127 integer format (i.e., add 127 to it)
- 23-bit to represent only the fractional part of the mantissa. The MSB of the mantissa is ALWAYS '1', therefore it is not stored

## Double Precision Floating Point number

- 64-bit double precision floating point:



$$x = -1^s \times 2^{exp-1023} \times 1.frac$$

$$2.2 \times 10^{-308} < |x| < 1.7 \times 10^{308}$$

- MSB is sign-bit (same as fixed point)
- 11-bit exponent in bias-1023 integer format (i.e., add 1023 to it)
- 52-bit to represent only the fractional part of the mantissa. The MSB of the mantissa is ALWAYS '1', therefore it is not stored

## Examples

### ■ Convert 5.75 to SP FP

- 5.75 to binary:  $+1.01110000... \times 2^2$
- exponent in bias-127 is  $127+2 = 129 = 1000\ 0000_b$
- The fractional part is  $.01110000...$  after we drop the hidden '1' bit.
- Answer:  $0\ 10000001\ 0111000\ 00...00 = 40B80000$  (hex)

### ■ Convert 0.1 to DP FP

- 0.1 to binary:  $1.10011001(1001\ \text{repeats}) \times 2^{-4}$
- exponent in bias-1023 is  $1023-4 = 1019 = 011\ 1111\ 1011_b$
- The fractional part is  $.10011001...1010$  after we drop the hidden '1' bit and rounding
- Answer:  $0\ 01111111011\ 1001100\ ...1001\ 1010 = 3FB9\ 9999\ 9999\ 999A$  (hex).

## Problems of Q-format

- Wrong Q-format representation will give totally wrong results
- Even correct use of Q-format notation may reduce precision
- For this example, Q12 result is totally wrong, and Q8 result is imprecise:

Q12	→ 7.50195		0111. 1000 0000 1000
Q12	→ 7.25	*	0111. 0100 0000 0000
Q24	→ 54.38916		0110 0110. 0110 0011 1010 0000 0000 0000
			Q12 → 6.38916
			Q8 → 54.38281

## Data types used by C6x DSPs

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	2's 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

## Special SP numbers

- IEEE floating point standard has a set of special numbers:

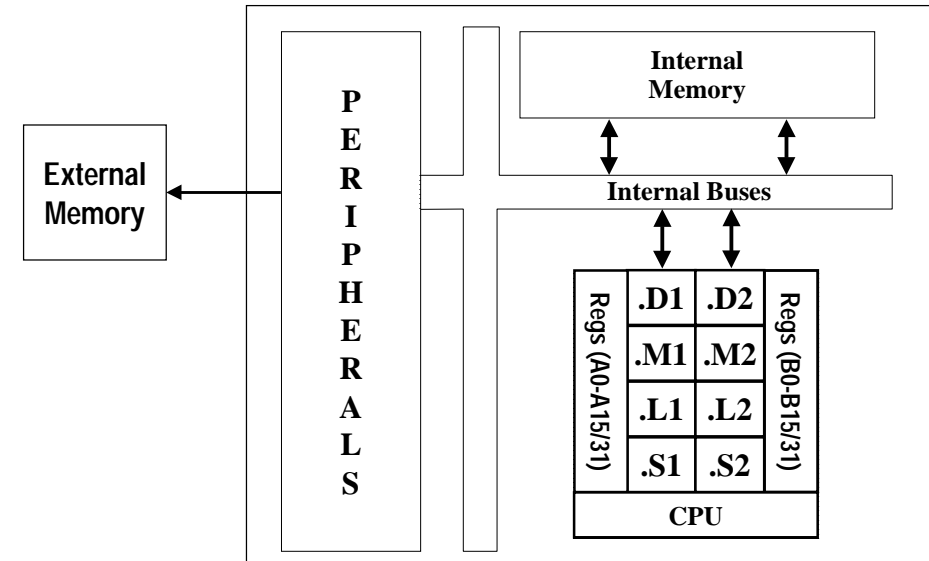
Special value	Sign (s)	Exponent (e)	Fraction (f)	Hex Value	Decimal value
+0	0	0	0	0x0000 0000	0.0
-0	1	0	0	0x8000 0000	-0.0
1	0	127	0	0x3F80 0000	1.0
2	0	128	0	0x4000 0000	2.0
+Inf	0	255	0	0x7F80 0000	$+\infty$
-Inf	1	255	0	0xFF80 0000	$-\infty$
NaN	x	255	Nonzero	0x7FFF FFFF	not a number
LFPN	0	254	All 1's	0x7F7F FFFF	$3.40282347 \times 10^{38}$
SFPN	0	1	All 0's	0x0080 0000	$1.17549435 \times 10^{-38}$

## Special DP numbers

■ Double precision floating point special numbers:

Special value	Exponent (e)	Fraction (f)	Hex Value	Decimal value
+0	0	0	0x0000 0000 0000 0000	0.0
-0	0	0	0x8000 0000 0000 0000	-0.0
1	1023	0	0x3FF0 0000 0000 0000	1.0
2	1024	0	0x4000 0000 0000 0000	2.0
+Inf	2047	0	0x7FF0 0000 0000 0000	+∞
-Inf	2047	0	0xFF0 0000 0000 0000	-∞
NaN	2047	Nonzero	0x7FFF FFFF FFFF FFFF	not a number
LFPN	2046	All 1's	0x7FEF FFFF FFFF FFFF	1.7976931348623157 e+308
SFPN	1	All 0's	0x0010 0000 0000 0000	2.2250738585072014 e-308

## TMS320C67x Internal System Architecture



## Four functional units for each datapath

Functional Unit	Fixed-Point Operations	Floating-Point Operations
.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations Leftmost 1 or 0 bit counting for 32 bits Normalization count for 32 and 40 bits 32-bit logical operations	Arithmetic operations Conversion operations: DP → SP, INT → DP, INT → SP
.S unit (.S1, .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from the control register file (.S2 only)	Compare reciprocal and reciprocal square-root operations Absolute value operations SP to DP conversion operations
.M unit (.M1, .M2)	16 × 16 bit multiply operations	32 × 32 bit multiply operations Floating-point multiply operations
.D unit (.D1, .D2)	32-bit add, subtract, linear and circular address calculation Loads and stores with a 5-bit constant offset Loads and stores with a 15-bit constant offset (.D2 only)	Load double word with a 5-bit constant offset

## Mapping of instructions to functional units

.S	.S Unit			.L	.L Unit		
	ADD	NEG	ABSSP		ABS	NOT	ADDSP
	ABDK	NOT	ABSDP		ADD	OR	ADDDP
	ADD2	OR	CMPTGSP		AND	SADD	SUBSP
	AND	SET	CMPEOSP		CMPEO	SAT	SUBDP
	B	SHL	CMPLTSP		CMPTG	SSUB	INTSP
	CLR	SHR	CMPTGDP		CMPLT	SUB	INTDP
	EXT	SSHI	CMPEODP		EMBD	SUBC	SPINT
	MV	SUB	CMPLTDP		MV	XOR	DPINT
	MVC	SUB2	RCPS		NEG	ZERO	SPRTUNC
	MVK	XOR	RCDP		NORM		DPTRUNC
	MVKH	ZERO	RSORSP				DPSP
			RSORDP				
			SPDP				
.L	.D Unit			.M Unit			
	ADD	NEG		MPY	SMPY	MPYSP	
	ADDAB (B/H/W)	STB (B/H/W)		MPYH	SMPYH	MPYDP	
	LDB (B/H/W)	SUB		MPYHL		MPYI	
LDDW	SUBAB (B/H/W)				MPYID		
MV	ZERO		No Unit Used				
			NOP		IDLE		

# Detailed internal datapaths

