Module 4

Introduction to Digital Signal (DS) Processors:

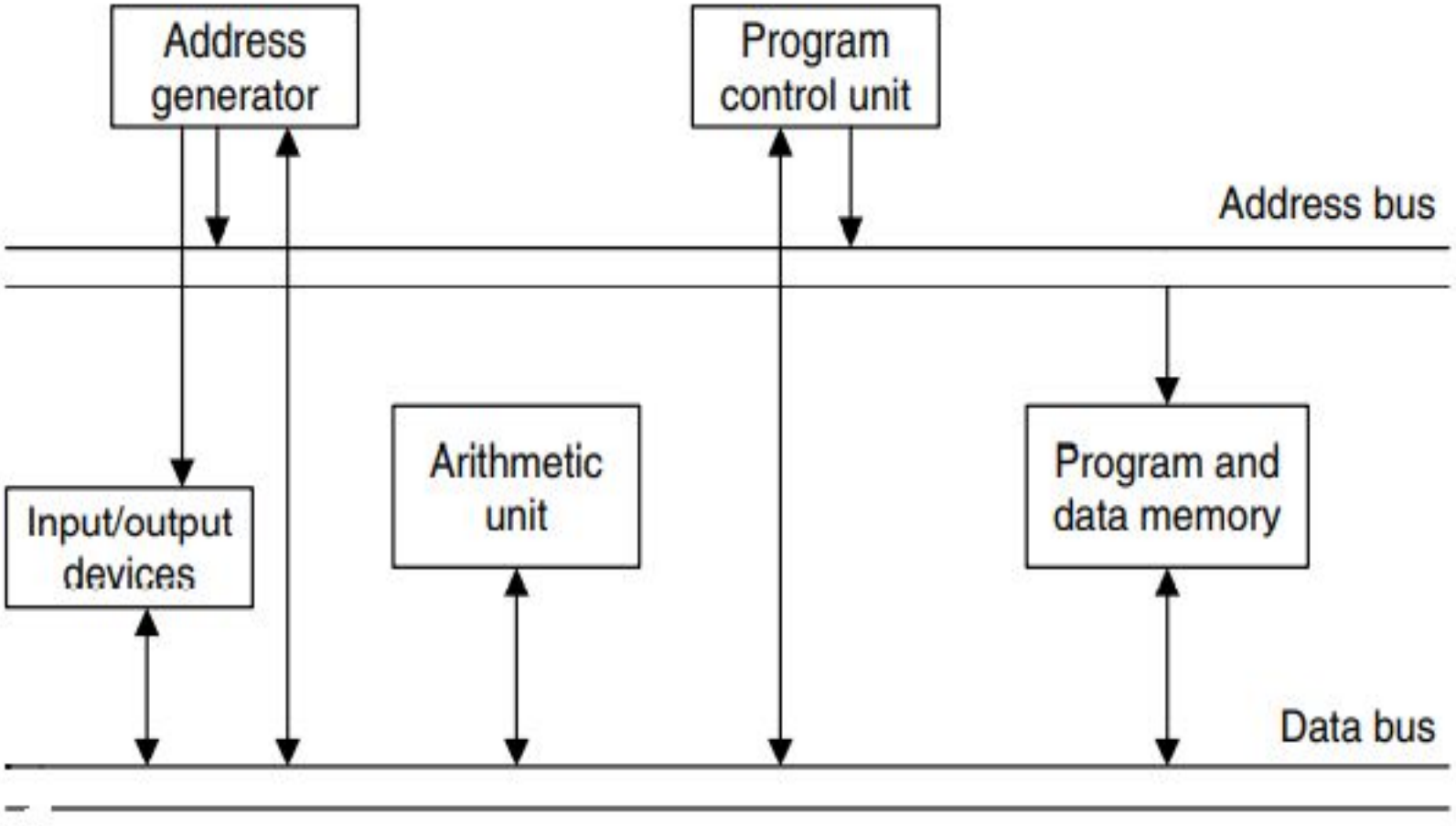
Introduction, Digital Signal Processor Architecture- Von Neumann architecture and Harvard architecture, Fixed- and Floating-Point Format for DS processors.

Text Book

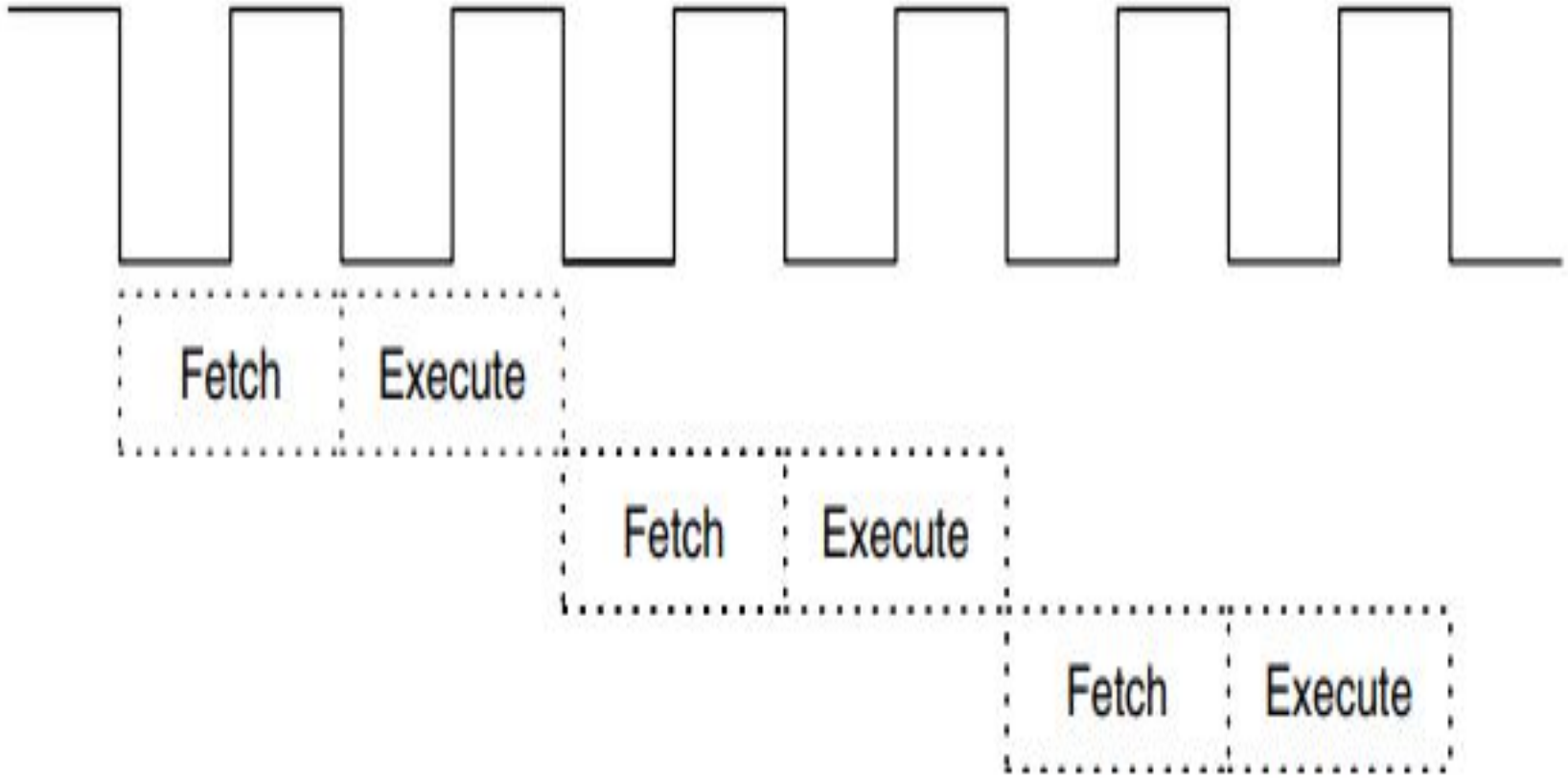
1.Proakis&Manolakis, “Digital signal processing – Principles Algorithms & Applications”, *Pearson education*, 4th Edition, New Delhi, 2007.

2. Li Tan, “Digital Signal Processing”, Academic Press, *Elsevier*, 2007.

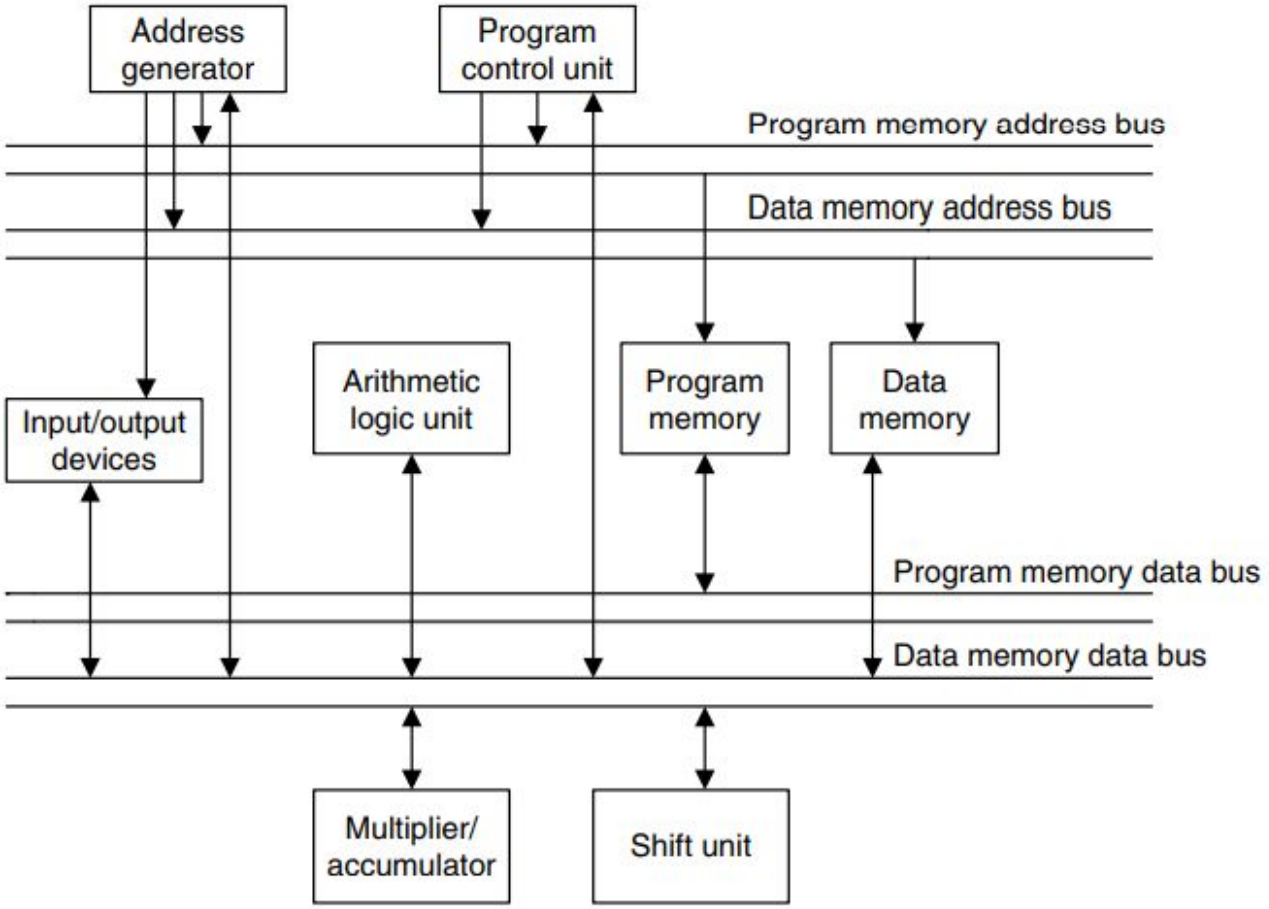
Von Neumann Architecture



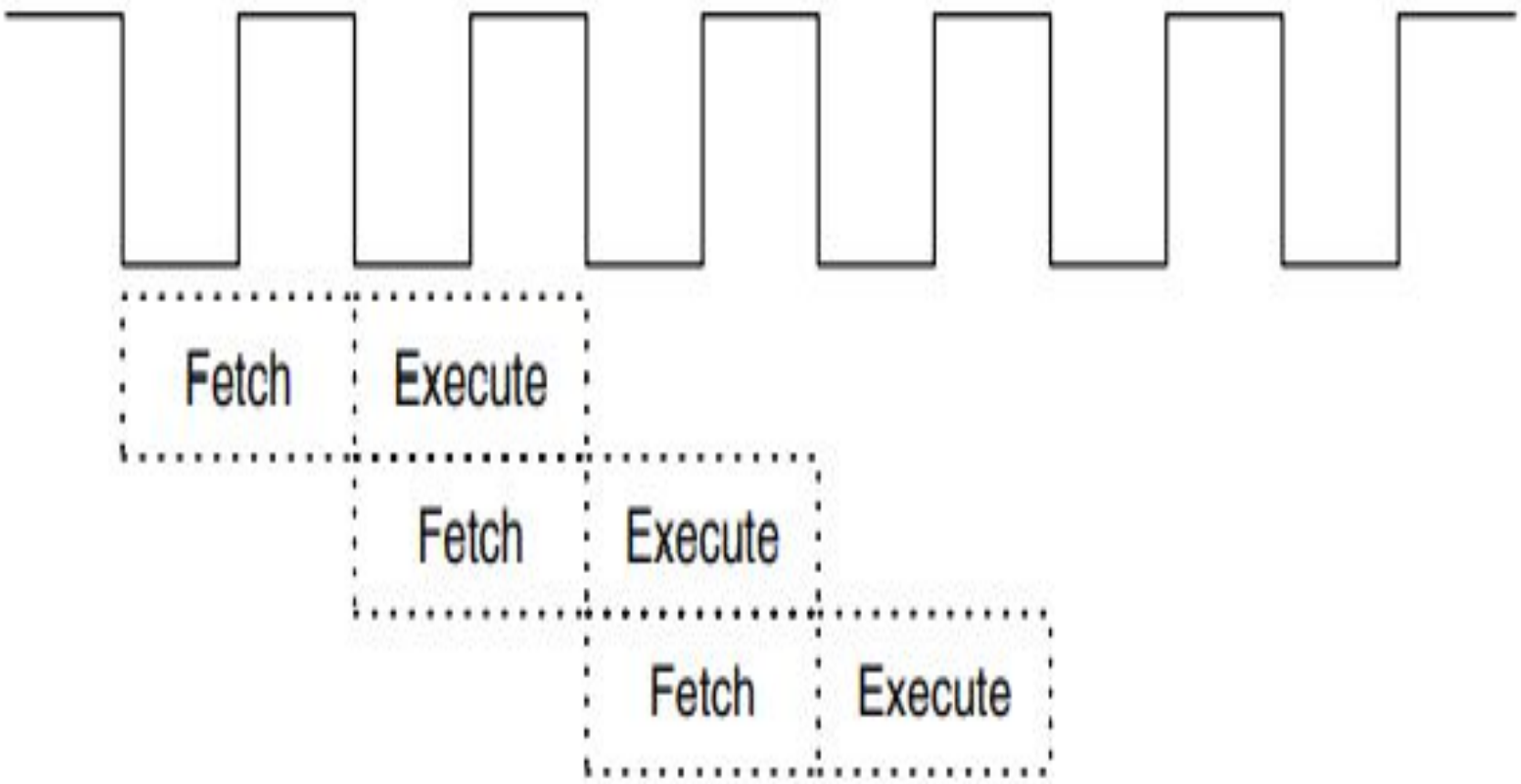
Execution cycle based on the Von Neumann Architecture



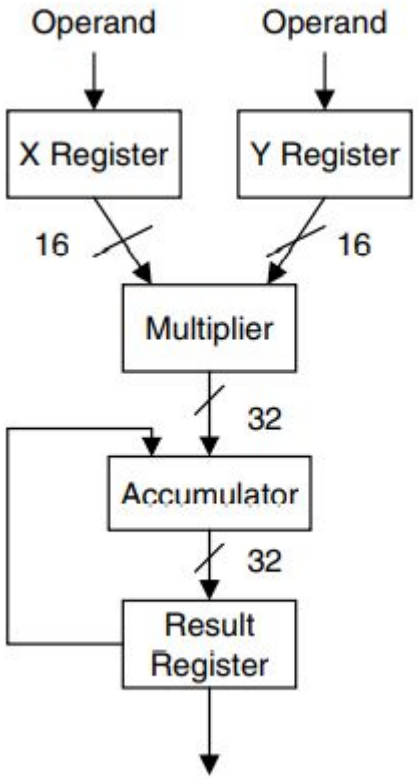
Harvard Architecture



Execution cycle based on the Harvard Architecture



Multiplier and Accumalator Unit(MAC) Dedicated to DSP



Fixed- and Floating-Point Format for DS processors

- Formats used by DSP implementation can be classified as fixed or floating point- s a fixed- or floating-point method depends on how the processor's CPU performs arithmetic.
- A fixed-point DS processor represents data in 2's complement integer format and manipulates data using integer arithmetic
- Disadv- Since operates using the integer format, which represents only a very narrow dynamic range of the integer number, a problem such as overflow of data manipulation may occur. Hence, we need to spend much more coding effort to deal with such a problem
- Adv- When it is time to make the DSP an application-specific integrated circuit (ASIC), a chip designed for a particular application, a dedicated hand-coded fixed-point implementation can be the best choice in terms of performance and small silica area
- A floating-point processor represents numbers using a mantissa (fractional part) and an exponent in addition to the integer format and operates data using floating-point arithmetic
- Adv- which offer a wider dynamic range of data, so that coding becomes much easier. However, the floating-point DS processor contains more hardware units to handle the integer arithmetic and the floating point arithmetic, hence is more expensive and slower than fixed-point processors in terms of instruction cycles.
- Therefore always choice is depending on the requirement

FIXED POINT ARITHMETIC

A 3-bit 2's complement number representation.

Decimal Number	2's Complement
3	011
2	010
1	001
0	000
−1	111
−2	110
−3	101
−4	100

2 X (-1)

```
   010
 × 001
-------
   010
   000
+ 000
-------
 00010
```

gives 110.

. Removing two extended sign bits

2's complement of 00010 = 11110.

Contd…

- 2 X (-3)

```
    010
  × 011
  -----
    010
   010
  000
  -----
  00110
```

2’s complement of 00110 = 11010. Removing two extended sign bits achieves 010. Since the binary number 010 is 2, which is not (-6) as we expect, overflow occurs; that is, the result of the multiplication (-(-6) is out of our dynamic range (-4 to 3)

Treating all decimals as fractional

A 3-bit 2's complement system using fractional representation.

Decimal Number	Decimal Fraction	2's Complement
3	3/4	0.11
2	2/4	0.10
1	1/4	0.01
0	0	0.00
−1	−1/4	1.11
−2	−2/4	1.10
−3	−3/4	1.01
−4	−4/4 = −1	1.00

A 3-bit 2's complement number representation.

Decimal Number	2's Complement
3	011
2	010
1	001
0	000
−1	111
−2	110
−3	101
−4	100

fractional binary 2's complement system.

The data are normalized to the fractional range from -1 to 1-2^-2=3/ 4.

When we carry out multiplications with two fractions, the result should be a fraction, so that multiplication overflow can be prevented.

$$
\begin{array}{r}
0.1\,0 \\
\times\ 0.1\,1 \\
\hline
0\,1\,0 \\
0\,1\,0 \\
+\ 0\,0\,0 \\
\hline
0.0\,1\,1\,0
\end{array}
$$

2's complement of 0.0110 = 1.1010.
The answer in decimal form should be

$$1.1010 = (-1) \times (0.0110)_2 = -\left(0 \times (2)^{-1} + 1 \times (2)^{-2} + 1 \times (2)^{-3} + 0 \times (2)^{-4}\right)$$

$$= -\frac{3}{8}.$$

This number is correct $\left(\frac{2}{4} \times \left(-\frac{3}{4}\right)\right) = -\frac{3}{8}$.

Additional error due to truncation

- If we truncate the last two least-significant bits to keep the 3-bit binary number, we have an approximated answer as
- $1.10 = (-1)x(0.10)_2 = -(1X2^{-1}+0X2^{-2}) = -1/2$

The truncation error occurs.

The error should be bounded by $2^{-2} = 1/4$.

We can verify that

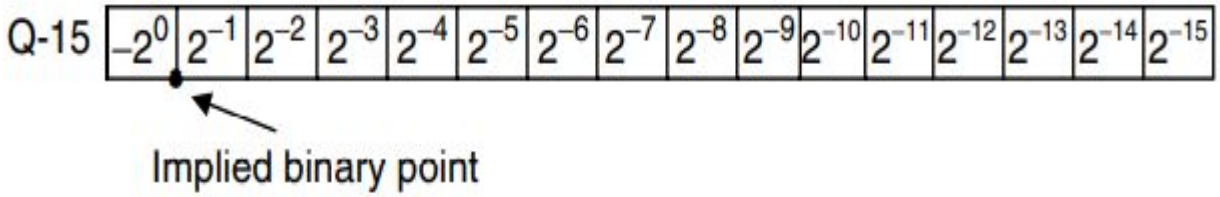
$|(-1/2)-(-3/8)| = 1/8 < 1/4$

Contd…

- Adding two positive fractional numbers yields a negative number. Hence, overflow occurs.

$$\begin{array}{r} 0.11 \\ +\ 0.01 \\ \hline 1.00 \end{array}$$

- Signed Q Format partially solves the overflow in multiplications. It is the most commonly used method in fixed point processor
- Q2-Format-2 magnitude bit and 1sign bit.
- In Q-15 format, 15 bits for magnitude and 1 bit for sign
- The number is normalized to the fractional range from -1 to 1. The range is divided into 2^16 intervals, each with a size of 2^-15. The most negative number is -1, while the most positive number is (1-2^-15). Any result from multiplication is within the fractional range of -1 to 1.



Number	Product	Carry
0.560123×2	1.120246	1 (MSB)
0.120246×2	0.240492	0
0.240492×2	0.480984	0
0.480984×2	0.961968	0
0.961968×2	1.923936	1
0.923936×2	1.847872	1
0.847872×2	1.695744	1
0.695744×2	1.391488	1
0.391488×2	0.782976	0
0.782976×2	1.565952	1
0.565952×2	1.131904	1
0.131904×2	0.263808	0
0.263808×2	0.527616	0
0.527616×2	1.055232	1
0.055232×2	0.110464	0 (LSB)

Contd…

- Since we use only 16 bits to represent the number, we may lose accuracy after conversion. Like quantization, the truncation error is introduced. However, this error should be less than the interval size, in this case,
- 2^-15 = 0.000030517.

Contd…

1.Find the Q-15 format of-0.160123.

- +0.160123 =0.001010001111110

2.Convert the Q-15 signed number 1.110101110000010 to the decimal number. Since the number is negative, applying the 2's complement yields

0.001010001111110:

0.160095

3. Convert the Q-15 signed number 0.100011110110010 to the decimal

0.560120

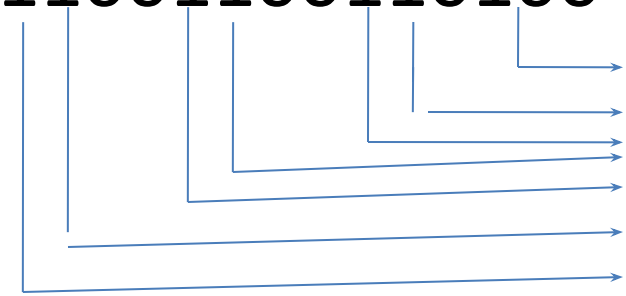
Contd…

Add 1.110101110000010 with

0. 100011110110010

Ans

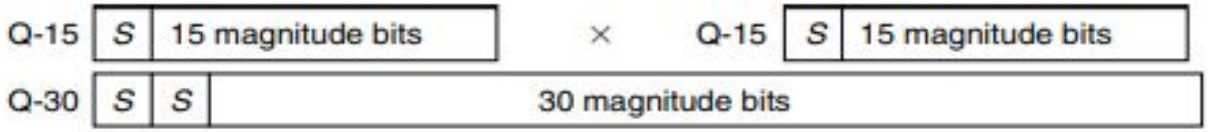
1 0. 011001100110100 :



Decimal equ is 0.400024:

Contd…

- The Q-format number representation is a better choice than the 2's complement integer representation because of no truncation error.
- But we need to be concerned with the following problems.
- 1. Converting a decimal number to its Q-N format, where N denotes the number of magnitude bits, we may lose accuracy due to the truncation error, which is bounded by the size of the interval, that is, 2^{-N}
- 2. Addition and subtraction may cause overflow, where adding two positive numbers leads to a negative number, or adding two negative numbers yields a positive number; similarly, subtracting a positive number from a negative number gives a positive number, while subtracting a negative number from a positive number results in a negative number.
- 3. Multiplying two numbers in Q-15 format will lead to a Q-30 format, which has 31 bits in total



- In practice, it is common for a DS processor to hold the multiplication result using a double word size such as MAC operation . In Q-30 format, there is one sign-extended bit. We may get rid of it by shifting left by one bit to obtain Q-31 format and maintaining the Q-31 format for each MAC operation.

Contd…

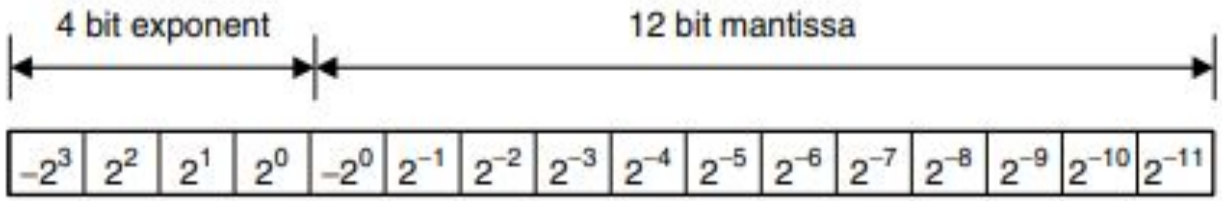
- Sometimes, the number in Q-31 format needs to be converted to Q-15; for example, the 32-bit data in the accumulator needs to be sent for 16-bit digital-to-analog conversion (DAC), where the upper most-significant 16 bits in the Q-31 format must be used to maintain accuracy.
- We can shift the number in Q-30 to the right by 15 bits or shift the Q-31 number to the right by 16 bits. The shifted result is stored in the lower 16-bit memory location. Note that after truncation, the maximum error is bounded by the interval size of 2^{-15}, which satisfies most applications.
- In using the Q format in the fixed-point DS processor, it is costlier to maintain the accuracy of data manipulation.
- 4. Underflow can happen when the result of multiplication is too small to be represented in the Q-format. As an example, in the Q-2 system, multiplying 0.01 with 0.01 leads to 0.0001. To keep the result in Q-2, we truncate the last two bits of 0.0001 to achieve 0.00, which is zero. Hence, underflow occurs.

Floating Point Format Representation

To increase the dynamic range of number representation and it is given by

$X=M \times 2^{E,}$ where M - mantissa or fractional part in Q format, E is the exponent.

The mantissa and exponent are signed numbers. If we assign 12 bits for the mantissa and 4 bits for the exponent,

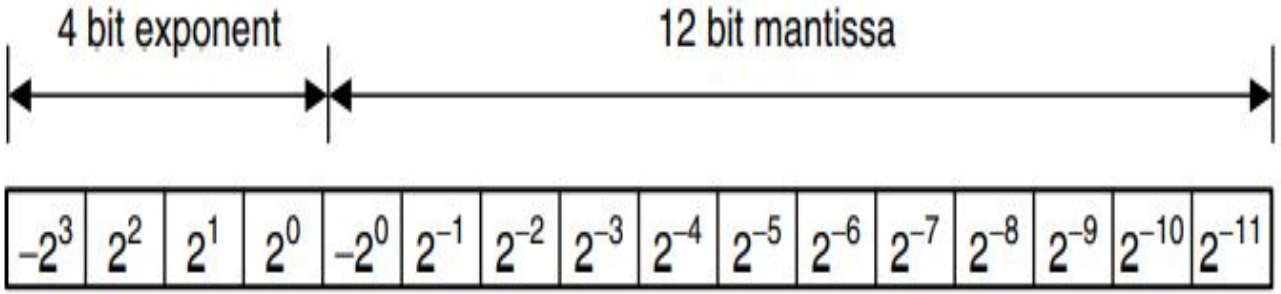


Since the 12-bit mantissa has limits between -1 and +1, the dynamic range is controlled by the number of bits assigned to the exponent. The bigger the number of bits assigned to the exponent, the larger the dynamic range.

The number of bits for the mantissa defines the interval in the normalized range;

for the above example the interval size is 2^{-11} in the normalized range, which is smaller than the Q-15. However, when more mantissa bits are used, the smaller interval size will be achieved.

Floating point representation



-

$$\text{Most negative number} = (1.00000000000)_2 \cdot 2^{0111_2} = (-1) \times 2^7 = -128.0$$

$$\text{Most positive number} = (0.11111111111)_2 \cdot 2^{0111_2} = \left(1 - 2^{-11}\right) \times 2^7 = 127.9375.$$

The smallest positive number is given by

$$\text{Smallest positive number} = (0.00000000001)_2 \cdot 2^{1000_2} = \left(2^{-11}\right) \times 2^{-8} = 2^{-19}.$$

Example

Convert each of the following decimal numbers to the floating-point number using the format specified in the previous slide

1. 0.1601230
2. 20.430527

Solution

a. satisfy the equality $0.5 \le M \le 1$

$$\text{scale the number } 0.1601230 \text{ to } 0.160123/2^{-2} = 0.640492$$

b. Represent in $M.2^E$ form

$$0.160123 = 0.640492 \times 2^{-2}$$

c. Determine binary equivalent of E and M concatenate both

$$-2 = 1110 \qquad 010100011111$$

$$1110010100011111$$

Solution contd…

$$-20.430527/2^5 = -0.638454.$$

$$-20.430527 = -0.638454 \times 2^5.$$

exponent bits should be 0101

0.638454 using Q-11 format gives:

010100011011.

−0.638454 as

101011100101.

Cascading the exponent bits and mantissa bits,

0101101011100101.

Floating point Arithmetic

- Rules for Arithmetic Addition
- If $x_1 = M_1 2^{E_1}$, $x_2 = M_2 2^{E_2}$.

$$x_1 + x_2 = \begin{cases} \left(M_1 + M_2 \times 2^{-(E_1 - E_2)}\right) \times 2^{E_1}, & \text{if } E_1 \geq E_2 \\ \left(M_1 \times 2^{-(E_2 - E_1)} + M_2\right) \times 2^{E_2} & \text{if } E_1 < E_2 \end{cases}$$

$$x_1 = M_1 2^{E_1}, \quad x_2 = M_2 2^{E_2}, \quad \text{where } 0.5 \leq |M_1| < 1 \text{ and } 0.5 \leq |M_2| < 1.$$

$$x_1 \times x_2 = (M_1 \times M_2) \times 2^{E_1 + E_2} = M \times 2^{E}$$

Floating point Arithmetic contd…

- Add two floating-point numbers Binary eq of 2 is =0010
- 1's comp 1101+
- 2's complement 1
- 1110 010100011111 = 0.640136718 X 2^{-2} 1110
- 0101 101011100101 = - 0.638183593 X 2^{5}
- E_1 is < E_2, To make them equal shift right M1 by 7 units
- 0.640136718= 0101 000000001010 = 0.005001068X 2^{5}
-

```
  0.0 0 0 0 0 0 0 0 1 0 1 0
+ 1.0 1 0 1 1 1 0 0 1 0 1
-------------------------
  1.0 1 0 1 1 1 0 1 1 1 1
```

the floating number as

0101 101011101111

Floating Point Arithmetic Contd…

Multiply two floating-point numbers

$$1110\ 010100011111 = 0.640136718 \times 2^{-2}$$
$$0101\ 101011100101 = -0.638183593 \times 2^{5}.$$

$$E_1 = 1110, E_2 = 0101, M_1 = 010100011111, M_2 = 101011100101.$$

Adding two exponents in 2's complement form leads to

$$E = E_1 + E_2 = 1110 + 0101 = 0011,$$

As per multiplication rule, only +ve values of mantissa need to be considered.
M1 is +ve but M2 –ve. So need to get 2's complement of M2= 010100011011.

010100011111 X 010100011011 = 001101000100 , taking 2's complement

M = 110010111100.
Therefore the **product is** obtained by cascading of exponent and M
0011 110010111100, whose decimal equivalent is
0.408203125 X 2^3 =-3.265625

Overflow in Floating Point Representation

- overflow will occur when a number is too large to be represented in the floating-point number system. Adding two mantissa numbers may lead to a number larger than 1 or less than -1; and multiplying two numbers causes the addition of their two exponents, so that the sum of the two exponents could overflow.

Case 1. Add the following two floating-point numbers:

0111 011000000000 + 0111 010000000000.

Note that two exponents are the same and they are the biggest positiv number in 4-bit 2's complement representation. We add two positiv mantissa numbers as

$$\begin{array}{rr} & 0.1100000000000 \\ + & 0.1000000000000 \\ \hline & 1.0100000000000. \end{array}$$

The result for adding mantissa numbers is negative. Hence, the over flow occurs.

Case 2. Multiply the following two numbers:

0111 011000000000 × 0111 011000000000.

Adding two positive exponents gives

0111 + 0111 = 1000 (negative; the overflow occurs).

Multiplying two mantissa numbers gives:

0.11000000000 × 0.11000000000 = 0.10010000000

Underflow in Floating Point Representation

underflow will occur when a number is too small to be represented in the number system. Let us divide the following two floating-point numbers:

$$1001\ 001000000000 \div 0111\ 010000000000.$$

First, subtracting two exponents leads to

$$1001\,(\text{negative}) - 0111\,(\text{positive}) = 1001 + 1001$$
$$= 0010\,(\text{positive; the underflow occurs}).$$

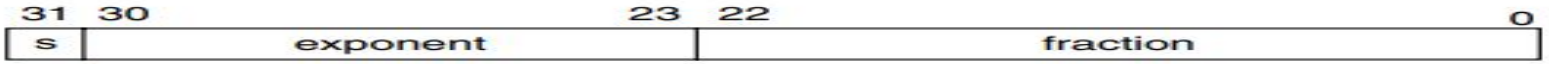
Then, dividing two mantissa numbers, it follows that

$$0.01000000000 \div 0.10000000000 = 0.10000000000$$

However, in this case, the expected resulting exponent is -14 in decimal, which is too small to be presented in the 4-bit 2's complement system. Hence the underflow occurs.

IEEE Floating Point Formats

- If $E = 255$ and F is nonzero, then $x = NaN$ ("Not a number").
- If $E = 255$, F is zero, and S is 1, then $x = -\text{Infinity}$.
- If $E = 255$, F is zero, and S is 0, then $x = +\text{Infinity}$.
- If $0 < E < 255$, then $x = (-1)^s \times (1.F) \times 2^{E-127}$, where $1.F$ represents the binary number created by prefixing F with an implicit leading 1 and a binary point.



$$x = (-1)^s \times (1.F) \times 2^{E-127}$$

23 fraction bits-Mantissa within the normalised range
Exponent E is in excess 127 form , value of 127 is the offset from the 8-bit exponent range from 0 to 255, so that E-127 will have a range from -127 to +128.

Convert the following number in the IEEE single precision format to the decimal format: 110000000.010 ... 0000

$$s = 1, E = 2^7 = 128$$
$$1.F = 1.01_2 = (2)^0 + (2)^{-2} = 1.25.$$

$$x = (-1)^1(1.25) \times 2^{128-127} = -1.25 \times 2^1 = -2.5.$$

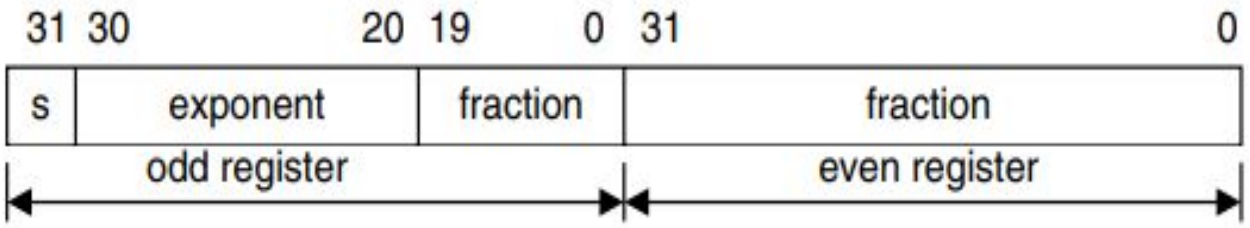
Contd…

- If $E = 0$ and F is nonzero, then $x = (-1)^s \times (0.F) \times 2^{-126}$. This is an "unnormalized" value.
- If $E = 0$, F is zero, and S is 1, then $x = -0$.
- If $E = 0$, F is zero, and S is 0, then $x = 0$.

Typical and exceptional examples are shown as follows:

$$0\ 00000000\ 00000000000000000000000 = 0$$
$$1\ 00000000\ 00000000000000000000000 = -0$$
$$0\ 11111111\ 00000000000000000000000 = \text{Infinity}$$
$$1\ 11111111\ 00000000000000000000000 = -\text{Infinity}$$
$$0\ 11111111\ 00000100000000000000000 = \text{NaN}$$
$$1\ 11111111\ 00100010001001010101010 = \text{NaN}$$
$$0\ 00000001\ 00000000000000000000000 = (-1)^0 \times (1.0_2) \times 2^{1-127} = 2^{-126}$$
$$0\ 00000000\ 10000000000000000000000 = (-1)^0 \times (0.1_2) \times 2^{0-126} = 2^{-127}$$
$$0\ 00000000\ 00000000000000000000001 =$$
$$(-1)^0 \times (0.00000000000000000000001_2) \times 2^{0-126} = 2^{-149} (\text{smallest positive value})$$

Double precision Format



64-bit word, which may be numbered from 0 to 63, left to right. The first bit is the sign bit *S*, the next eleven bits are the exponent bits *E*, and the final 52 bits are the fraction bits *F*. The IEEE floating-point format in double precision significantly increases the dynamic range of number representation, since there are eleven exponent bits; the double precision format also reduces the interval size in the mantissa normalized range of +1 to +2, since there are 52 mantissa bits as compared with the single precision case of 23 bits.

- Convert the following number in IEEE double precision format to the decimal format:

  001000 ... 0.110 ... 0000 using the above bit pattern

$$s = 0, E = 2^9 = 512 \text{ and}$$
$$1.F = 1.11_2 = (2)^0 + (2)^{-1} + (2)^{-2} = 1.75$$

$$x = (-1)^0(1.75) \times 2^{512-1023} = 1.75 \times 2^{-511} = 2.6104 \times 10^{-154}.$$

If $E = 2047$ and F is nonzero, then $x = NaN$ ("Not a number")

If $E = 2047$, F is zero, and S is 1, then $x = -\text{Infinity}$

If $E = 2047$, F is zero, and S is 0, then $x = +\text{Infinity}$

If $0 < E < 2047$, then $x = (-1)^s \times (1.\text{F}) \times 2^{E-1023}$, where $1.F$ is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point

If $E = 0$ and F is nonzero, then $x = (-1)^s \times (0.F) \times 2^{-1022}$. This is an "unnormalized" value

If $E = 0$, F is zero, and S is 1, then $x = -0$

If $E = 0$, F is zero, and S is 0, then $x = 0$