CSE340: Computer Architecture

Handout_Chapter - 3: Arithmetic for Computers

BRAC
UNIVERSITY

Inspiring Excellence

Prepared by: Partha Bhoumik (PBK)
Course Coordinator, CSE340

Integer Addition-Subtraction

\# Addition: $(7)_{10} + (6)_{10}$

Bits with this color represent carry-forward

$$
\begin{array}{cccccc}
 & 1 & 1 & 0 & & \\
 & 0 & 1 & 1 & 1 & \\
+ & 0 & 1 & 1 & 0 & \\
\hline
 & 1\ (1) & 1\ (1) & 0\ (0) & 1 &
\end{array}
$$

\# adding bits right to left.

\# Subtraction: $(7)_{10} - (6)_{10} = (7)_{10} + (-6)_{10}$

$$
\begin{aligned}
6 &= 110 \\
+6 &= 0110 \\
&= 1001 \\
&\quad\ +1 \\
\hline
-6 &= (1010)_{2}
\end{aligned}
$$

$$
\begin{array}{ccccc}
 & 1 & 1 & 1 & 0 \\
+7 = & 0 & 1 & 1 & 1 \\
-6 = & 1 & 0 & 1 & 0 \\
\hline
 & 1\ (1) & 0\ (1) & 0\ (1) & 0\ (0)\ 1
\end{array}
$$

Overflow Detection (integer)

#Addition:

Case-1: Add two same signed numbers:
if (answer also has same sign):
    No overflow
else:
    Overflow
Case-2: Add two different signed numbers:
    Never Overflow

#Subtraction:

Case-3: Sub two same signed numbers:
    Never Overflow
Case-4: Sub two different signed numbers:
    +A - (-B) = +A +B => Case-1
    -A - (+B) = -A +(-B) => Case-1

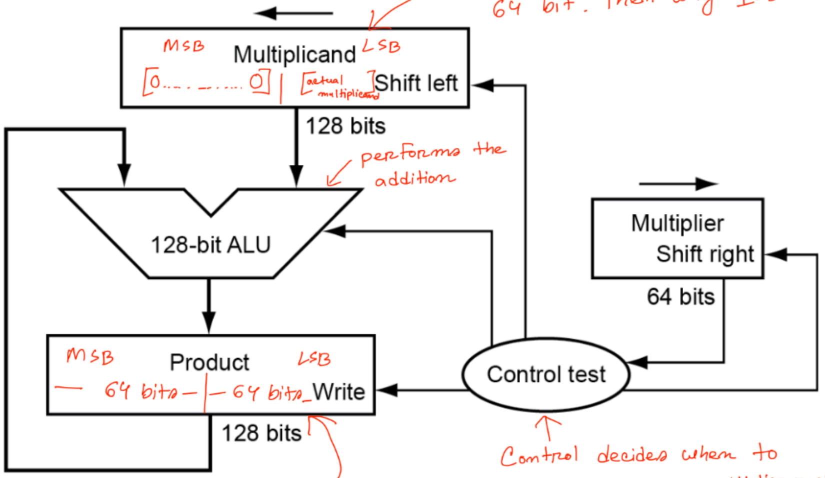
Long Multiplication

```
      1000  ——> Multiplicand
   x  1001  ——> Multiplier
   ---------
      1000
     0000
    0000
   1000
   ---------
   1001000  ——> Product
```

Maximum,
length of the product
= (length of multiplicand
+ " " multiplier)

System:

this was also supposed to be 64 bit. Then why 128 bits?



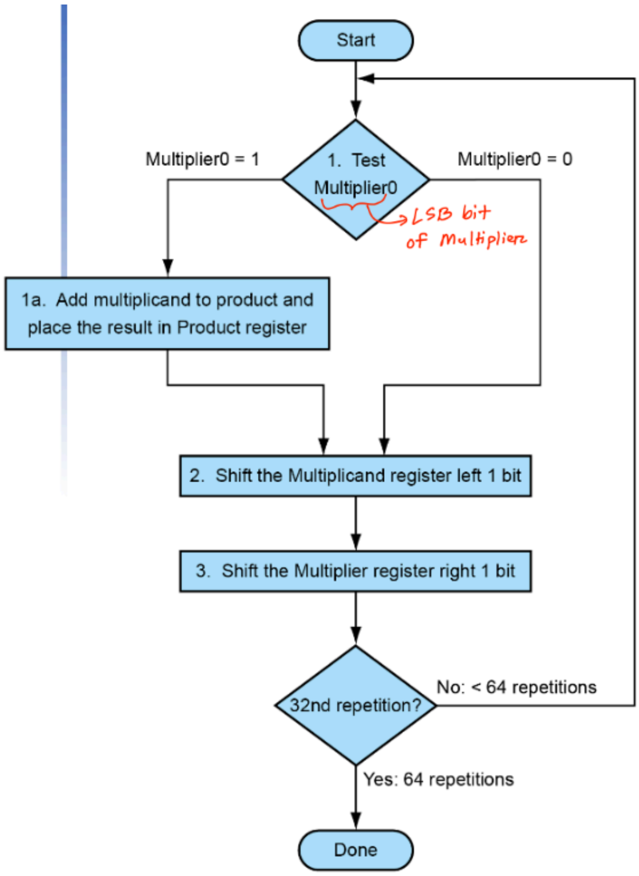
Control decides when to shift multiplicand, multiplier register, when to use ALU, write data in product.

Multiply two 64 bit values, product length can be (64+64)=128 bit.

But we do not have any 128 bit registers in RISC-V

Hence, we use two registers to store the values

Flowchart:



Example:

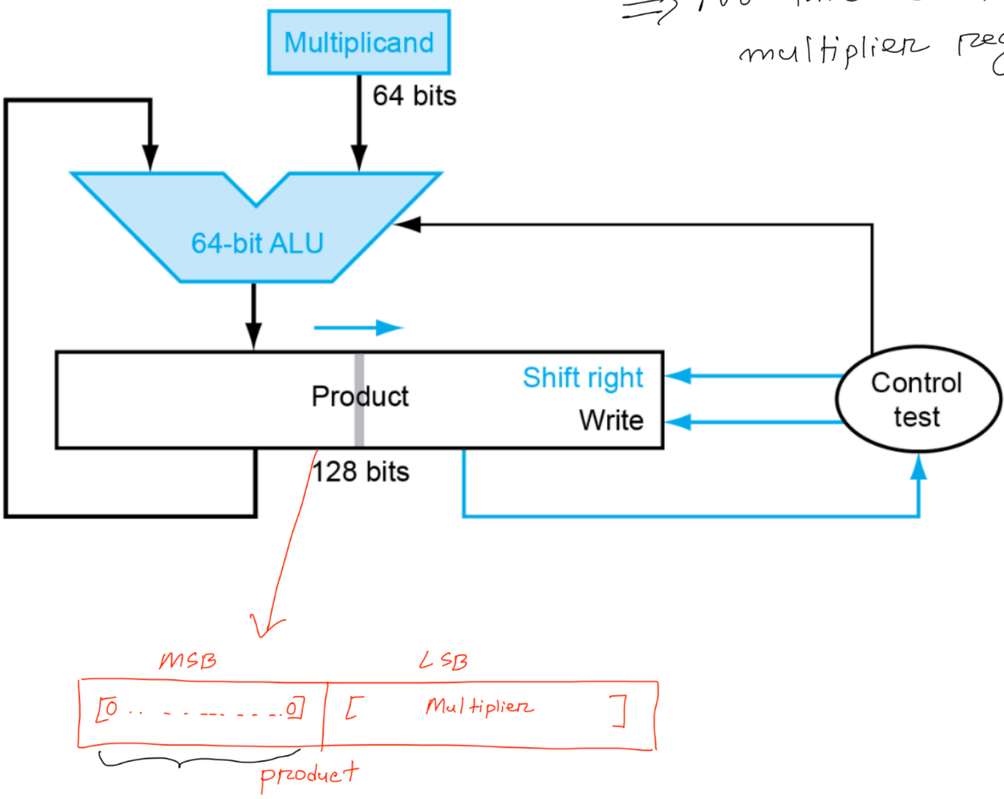
Multiply 8 and 9 using the long multiplication method:
Solution:
You can choose any of the operands as multiplier or multiplicand.
Number of iterations = Number of bits in Multiplier

Iteration	Multiplier	Multiplicand	Product
0	1001	0000 1000	0000 0000
1	1001	0000 1000	0000 1000
	1001	0001 0000	0000 1000
	0100	0001 0000	0000 1000
2	0100	0010 0000	0000 1000
	0010	0010 0000	0000 1000
3	0010	0100 0000	0000 1000
	0001	0100 0000	0000 1000
4	0001	0100 0000	0100 1000
	0001	1000 0000	0100 1000
	0000	1000 0000	0100 1000

Optimized Multiplication

System:

⇒ No individual multiplier reg.



Number of iteration = Number of bits in Multiplier

Logic:

```
if (iteration <= multiplier bit length):
        if (multiplier0 == 1):
                product_MSB = Multiplicand + product_MSB
                product = right shift product by 1
        elif (multiplier0 == 0):
                product = right shift product by 1
```

Example:

Multiply 8 and 9 using the optimized multiplication method:
Solution:
You can choose any of the operands as multiplier or multiplicand.
Number of iterations = Number of bits in Multiplier
This color represents the product MSB part.

Iteration	Multiplicand	Product
0	1000	0000 1001
1	1000	1000 1001
		0100 0100
2	1000	0010 0010
3	1000	0001 0001
4	1000	1001 0001
		0100 1000

Floating Point

How does RISC-V support numbers with fractions?
=> Using IEEE-754 floating point representation

Scientific Notation is just a way to represent very large or very small number.

$\Rightarrow 4500000 = \underbrace{4.5}_{\text{Coefficient}} \times \underbrace{10}_{\text{Base}}^{\overbrace{6}^{\text{Exponent}}}$

$\Rightarrow 0.00453 = 4.53 \times 10^{-3}$

Decimal

$\Rightarrow 5.64 \times 10^{33}$
$\Rightarrow -2.34 \times 10^{56}$
→ Normalized.

$\Rightarrow 109.64 \times 10^{33}$
$\Rightarrow 0.002 \times 10^{-4}$
$\Rightarrow +987.02 \times 10^{9}$
→ Not Normalized.

In Binary,

$\pm 1.xxxx_2 \times 2^{yyy}$

IEEE-754 floating point representation:
i. Single Precision. (32 bits)
ii. Double Precision. (64 bits)

Using double precision, you can represent a larger or a smaller number than single precision.

Normalized Number

Binary Point (representing Binary Numbers)

⇒ A binary number is Normalized if :

i) Only one digit before the binary point.

ii) And that digit must be a non-zero number.

⇒ 11.00101×2^{35} ✗ not a normalized number.

⇒ 1.100101×2^{37} ✓ a normalized number.

\# To normalize a number you need to shift the binary point (.) left or right until you have a single non-zero digit before the binary point.

⇒ If you shift left, the number of times you left shifted will be added with the exponent.

$\Rightarrow 110.111 \times 2^{35}$

$\Rightarrow 1.10111 \times 2^{35+2}$

⇒ If you shift right, the number of times you right shifted will be subtracted from the exponent.

$\Rightarrow 0.00110$

$\Rightarrow 0.00110 \times 2^{0}$

$\Rightarrow 1.10 \times 2^{0-3} = 1.10 \times 2^{-3}$

IEEE-754 Floating Point Representation

	Sign Bit	Exponent	Fraction
Single P.	1 bit	8 bits	23 bits
Double P.	1 "	11 "	52 "

IEEE-754 Single Precision Format (32-bit)

Sign Bit	Exponent (Biased)	Fraction
1	8	23

Sign Bit = 0 ⇒ positive number

1 ⇒ negative "

Exponent = It will be represented as unsigned number.

normalized

1.1011×2^{34} → Exponent

1.1011×2^{-8}

8 bit unsigned binary range $= 0$ to $2^8 - 1$

$= 0$ to 255

But 0000 0000 and 1111 1111 are reserved, so the range for biased exponent is 1 to 254

If the size of biased exponent field is n bits, $\text{Bias} = 2^{(n-1)} - 1$

Hence, for 8 bit biased exponent, bias $= 2^7 - 1 = 127$

IEEE-754 Double Precision Format (64-bit)

Sign Bit	Exponent (Biased)	Fraction
1	11	52

Sign Bit = 0 $\Rightarrow$ positive number

1 $\Rightarrow$ negative "

Exponent = It will be represented as unsigned number.

11 bit unsigned binary range $= 0$ to $2^{11}-1$

$= 0$ to 2047

But 000 0000 0000 and 111 1111 1111 are reserved, so the range for biased exponent is 1 to 2046

If the size of biased exponent field is n bits, Bias $= 2^{(n-1)}-1$

Hence, for 11 bit biased exponent, bias $= 2^{10}-1 = 1023$

Decimal to IEEE-754 Floating Point Conversion

Steps:
1. Convert the decimal number to binary number.
2. Normalize the binary number.
3. Find the biased exponent.
4. Sign Bit.
5. Find the fraction.
6. Encode accordingly.

Example:

Convert 50.6749 to IEEE-754 single precision floating point representation. Show your final answer in Hex format. Consider 10 bits while converting from decimal to binary.

Solution:

50 = 11 0010

·6749 = 10 1011 0011....

(i) 50.6749 = (11 0010 . 10 1011 0011 ....)

= 11 0010 . 10 1011 0011 .... $\times 2^0$

(ii) = 1. 1001 0101 0110 011... $\times 2^5$ ← actual exponent

(iii) Bias = $2^{8-1} - 1 = 2^7 - 1 = 127$

∴ Biased exponent = 5 + 127 = 132 = 1000 0100

(iv) positive number; sign bit = 0

(v) $1.\underbrace{1001\ 0101\ 0110\ 011...}_{\text{fraction}} \times 2^5$

Fraction = 1001 0101 0110 011 00000000

rest of the bits will be filled up by 0s.

·6749 × 2 = 1.3498
·3498 × 2 = 0.6996
·6996 × 2 = 1.3992
·3992 × 2 = 0.7984
·7984 × 2 = 1.5968
·5968 × 2 = 1.1936
·1936 × 2 = 0.3872
·3872 × 2 = 0.7744
·7744 × 2 = 1.5488
·5488 × 2 = 1.0976

Sign Bit	Exponent (Biased)	Fraction
0	1000 0100	1001 0101 0110 011 00000000

50.6749 = 0100 0010 0100 1010 1011 0011 0000 0000

= 0x424AB300

Example:

Convert -0.0232 to 12-bit IEEE-754 representation where the size of the exponent field is 4 bits. Show your final answer in Hex format.

Solution:

(i) $-0.0232 = -0.0000010$

(ii) $-0.0000010 = 1.0 \times 2^{-6}$ ← actual exponent

fraction

(iii) $\text{Bias} = 2^{4-1} - 1 = 7$

$\therefore$ Biased Exponent $= -6 + 7 = 1 = 0001$

(iv) Sign Bit $= 1$.

(v) Fraction $= 000\ 0000$

1	0001	000 0000

$-0.000232 = 1000\ 1000\ 0000$

$= 0x880$ (Ans)

IEEE-754 Floating Point to Decimal Conversion

Steps:

1. Convert the Hex/Decimal number to binary number.
2. Arrange the binary number according to the given IEEE format.
3. Determine the sign.
4. Find out the actual exponent from the biased exponent field.
5. Convert Fraction to Decimal.
6. Final number = $(-\ 1)^{Sign\ Bit} \times (1\ +\ Fraction) \times 2^{Actual\ Exponent}$.

Example:

Convert the given IEEE-754 single precision floating point number 0xF2400120 to decimal.

Solution:

(i) 1111 0010 0100 0000 0000 0001 0010 0000

(ii) $\underset{\substack{\uparrow \\ \text{Sign Bit}}}{1}\ \underbrace{111\ 00100}_{\substack{\uparrow \\ \text{Biased Exp.}}}\ \underbrace{100\ 0000\ 0000\ 0001\ 0010\ 0000}_{\text{Fraction}}$

(iii) sign = −

(iv) Biased exp. = 111 0010 0 = 228

Bias = $2^{8-1} - 1 = 127$

$\therefore$ Exponent = 228 − 127 = 101

(v) Fraction = 100 0000 0000 0001 0010 0000

= 0.100 0000 0000 0001 0010 0000

= 0.5000343323

(vi) Decimal Value = $(-1)^1 \times (1 + 0.5000343323) \times 2^{101}$

$= -1.5000343323 \times 2^{101}$

Floating Point Addition/Subtraction

Given A and B are both floating-point numbers.
Steps:
1. Make sure both numbers are in Binary.
2. Normalize both A and B.
3. Align the binary point so that the lower exponent matches with the higher exponent.
4. Now add or sub accordingly.
5. Normalize the result.

Ex: $9.999 \times 10^{1} + 1.610 \times 10^{-1}$; size of exponent field is 3 bits

$= 99.99 + 0.1610$

$= 1100011.1111\ 1101\ 01 + 0.0010\ 1001\ 00$

$= 1.1000\ 1111\ 1111\ 0101 \times 2^{6} + 1.0100\ 100 \times 2^{-3}$

$= 1.1000\ 1111\ 1111\ 0101 \times 2^{6} + 0.0000\ 0000\ 1\ 0100\ 100 \times 2^{6}$

$= 1.10010 \times 2^{6}$ (Ans)

Bias $= 2^{3-1} - 1 = 3$

Biased Exp. $= 3 + 6 = 9$

Range $= 0$ to $2^{3} - 1$

$= 0$ to 7

$= 1$ to 6 [reserved 0 and 7]

upper range

$9 > 6 \Rightarrow$ So, overflow

\# $110100.111011 \times 2^{8} + 10110.11111 \times 2^{7}$

$= 1.1010\ 0111\ 011 \times 2^{13} + 1.0110\ 1111\ 1 \times 2^{11}$

$= 1.1010\ 0111\ 011 \times 2^{13} + 0.0101\ 1011\ 11 \times 2^{13}$

$= 10.0000\ 0011\ 010 \times 2^{13}$

$= 1.0000\ 0001\ 1010 \times 2^{14}$ (Ans)

Floating Point Multiplication

Given A and B are both floating-point numbers.
Steps:
1. Make sure both numbers are in Binary.
2. Normalize both A and B.
3. Add the exponents.
4. Now multiply accordingly.
5. Normalize the result.
6. Determine the sign of the operation.

<u>Ex:</u> $1.110 \times 2^{5} \times 1.11 \times 2^{-5}$

$= 1.110 \times 1.11 \times 2^{5+(-5)}$

$= 11.0001 \times 2^{0}$

$= 1.10001 \times 2^{1}$ (Ans)

Overflow-Underflow detection for IEEE-754 format

Overflow / Underflow detection:

Given number is too small to represent using the mentioned system.

Step 1: Find the biased exponent of the answer.

Step 2: " " range of the biased exponent of the given system. (1 to upper Range)

Step 3: Detection:

if (Biased exponent < 1):

underflow

else if (Biased exponent > upper Range)

overflow

else : [1 ≤ Biased Exp ≤ upper Range]

No over/under flow

Floating Point instructions in RISC-V

Suppose, two single Preci. floating point numbers A, B are stored in memory. The memory locations are directly stored in register x_{10}, x_{11}. Write necessary code to store the result of A+B in the memory address that is stored in x_{13}.

Sol$^{\underline{n}}$:

```
flw     f1, 0(x10)    ; f1 = A
flw     f2, 0(x11)    ; f2 = B
fadd.s  f3, f1, f2    ; f3 = A+B
fsw     f3, 0(x13)
```

Formulas

1. $Bias = 2^{(n-1)} - 1\,;\ n = Size\ of\ the\ exponent\ field$

2. $Biased\ Exponent = Bias + Actual\ Exponent$
   $Actual\ Exponent = Biased\ Exoponent - Bias$

3. $Biased\ Exponent\ Range = 0\ to\ 2^n - 1\,;\ [upper\ \&\ lower\ limit\ is\ reserved]$
   $= 1\ to\ 2^n - 2\,;\ [Usable\ range]$

4. $Actual\ Exponent\ Ramge = (0 - Bias)\ to\ (2^n - 1 - Bias)$
   $= (1 - Bias)\ to\ (2^n - 2 - Bias)\,;\ [Usable\ range]$

5. $Decimal\ Number = (-1)^{Sign\ Bit} \times (1 + Fraction) \times 2^{Actual\ Exponent}$