

Integer & Fixed Point Addition and Multiplication

CENG 329 Lab Notes

By F. Serdar TAŞEL

Integers

- Generally we use 8-bits, 16-bits, 32-bits or 64-bits to store integers.
- $20 = 16+4 = (0001\ 0100)_2$ 8-bits
- $20 = 16+4 = (0000\ 0000\ 0001\ 0100)_2$ 16-bits
←padded with zeros→
- We use 2's complement format for the notation of negative signed numbers:

$$20 = (0\dots01\ 0100)_2$$

$$-20 = (1110\ 1100)_2 \quad 8\text{-bits}$$

$$-20 = (1111\ 1111\ 1110\ 1100)_2 \quad 16\text{-bits}$$

←padded with ones→

Sign bit

Integers

- How to store integers in registers?
- Consider that we have 8-bit registers.
- $20 = (10100)_2$
- As 8-bit integer: (r1)
 - $r1 = 20 = (0001\ 0100)_2$
- As 16-bit integer: (r1 r2)
 - $r1 = 0 = (0000\ 0000)_2$
 - $r2 = 20 = (0001\ 0100)_2$
 - $(r1\ r2) = 20 = (0000\ 0000\ 0001\ 0100)_2$
- As 32-bit integer: (r1 r2 r3 r4)
 - $r1 = r2 = r3 = 0 = (0000\ 0000)_2$
 - $r4 = 20 = (0001\ 0100)_2$
 - $(r1\ r2\ r3\ r4) = 20 = (0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0100)_2$

Integers

- Represent 123456789 in 32-bit integer:
 - $123456789 = (111\ 0101\ 1011\ 1100\ 1101\ 0001\ 0101)_2$
 - Convert to 32-bits:
 - $0000\ 0111\ 0101\ 1011\ 1100\ 1101\ 0001\ 0101$
 - $r1 = (0000\ 0111)_2 = 0x07 = 7$
 - $r2 = (0101\ 1011)_2 = 0x5b = 91$
 - $r3 = (1100\ 1101)_2 = 0xcd = 205$
 - $r4 = (0001\ 0101)_2 = 0x15 = 21$
 - $(r1\ r2\ r3\ r4) = 0x075bcd15$
 $= (0000\ 0111\ 0101\ 1011\ 1100\ 1101\ 0001\ 0101)_2$
 $= 123456789$

Integers

- Given following values of registers, find the value of (r4 r3 r2 r1)?
- $r1 = 72$, $r2 = 100$, $r3 = 250$, $r4 = 255$

$$r1 = 72 = (0100\ 1000)_2$$

$$r2 = 100 = (0110\ 0100)_2$$

$$r3 = 250 = (1111\ 1010)_2$$

$$r4 = 255 = (1111\ 1111)_2$$

$$(r4\ r3\ r2\ r1) = (1111\ 1111\ 1111\ 1010\ 0110\ 0100\ 0100\ 1000)_2$$

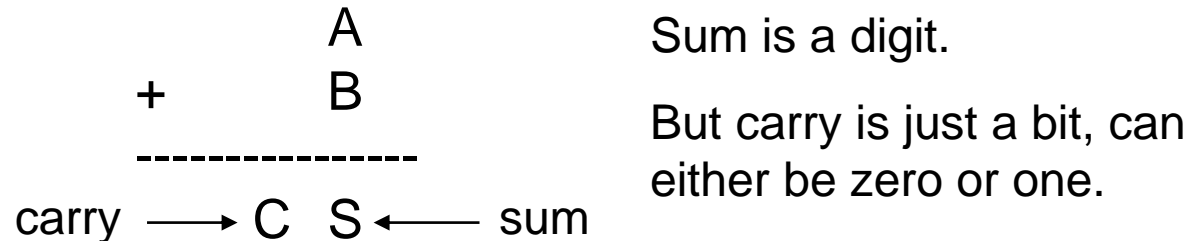
The number is negative! Take 2's complement:

$$(0000\ 0000\ 0000\ 0101\ 1001\ 1011\ 1011\ 1000)_2$$

$$= -367544$$

Integer Additon

- Assume that you have an operator that adds only two digits:



Each digit is a number in a base b.

b=10 \Rightarrow numbers: 0-9

b=2 \Rightarrow numbers: 0-1

b=2⁸=256 \Rightarrow numbers: 0-255

Operator:

← Addition table!

← AND/XOR Operator

← ADD for Intel(x86)
or Zilog

Note that the sum of two single-digit yields one digit and extra one bit at most!

Integer Additon

- Assume that we have an operator that adds only two digit. How can we add two numbers with multiple digits?

$$\begin{array}{r} 5639 \\ + 1427 \\ \hline \end{array}$$

?

Solution: Add digits individually
 Also add carry!

Integer Additon

$$\begin{array}{r} 5639 \\ + 1427 \\ \hline 6 \text{ (Carry=1)} \end{array}$$

Integer Additon

$$\begin{array}{r} 1 \\ 56\boxed{3}9 \\ + 14\boxed{2}7 \\ \hline 66 \text{ (Carry=0)} \end{array}$$

Integer Additon

$$\begin{array}{r} 0 \\ 5639 \\ + 1427 \\ \hline 066 \text{ (Carry=1)} \end{array}$$

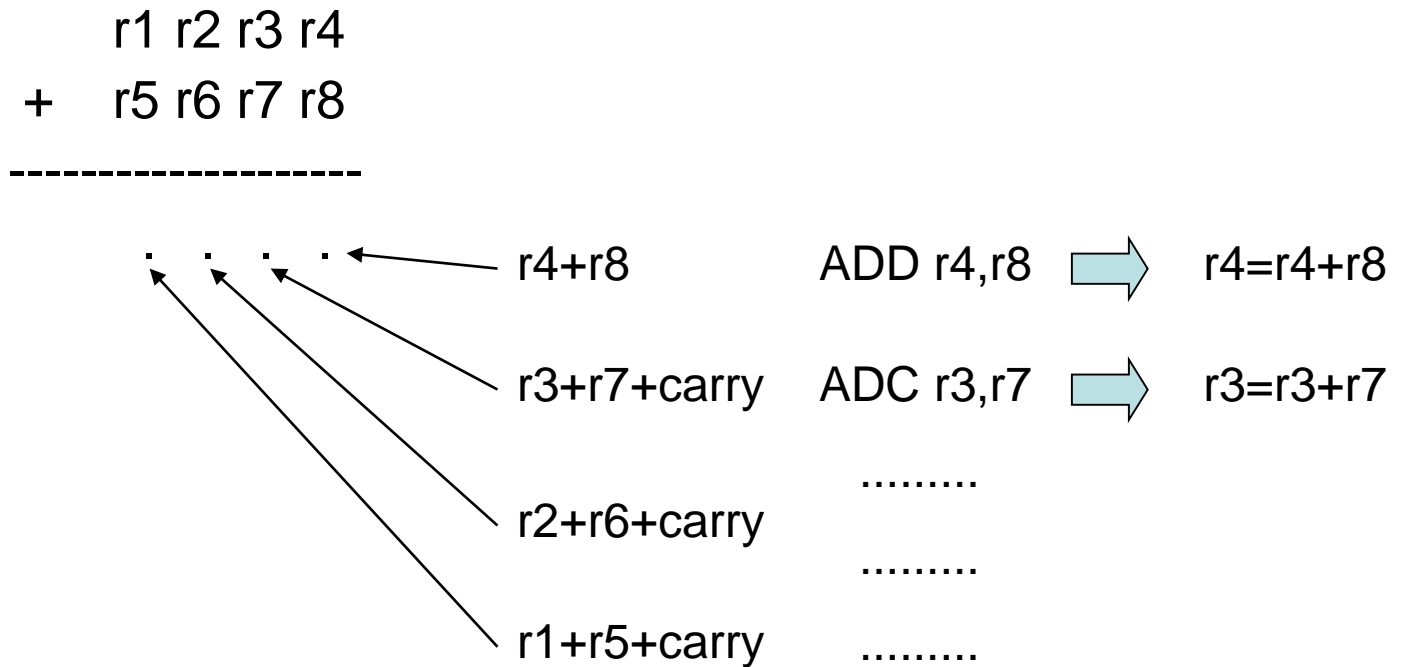
Integer Additon

$$\begin{array}{r} 1 \\ 5639 \\ + 1427 \\ \hline 7066 \text{ (Carry=0)} \end{array}$$

Integer Additon

Now consider that we are working in base 256.

Put each digit in a register so that we'll have 4 register for each 32-bit number.



Integer Additon

- What about signed numbers?
- Use 2's complement for negative numbers and just add! Ignore the last produced carry.
- How does it work? Explained later...
- What about subtraction?
- Subtraction can easily be implemented by taking 2's complement of the second operand first and then applying addition:
 - $A - B = A + (-B)$

Integer Multiplication

- Assume that you have an operator that multiplies only two digits:

$$\begin{array}{r} \\ x B \\ \hline C \end{array}$$

Each digit is a number in a base b .

$b=10 \Rightarrow$ numbers: 0-9

$b=2 \Rightarrow$ numbers: 0-1

$b=2^8=256 \Rightarrow$ numbers: 0-255

Operator:

Times table!

AND Operator

MUL for Intel(x86)

MULT for Zilog

Note that the product of two single-digit yields two digits at most!

Integer Multiplication

- Assume that we have an operator that can multiply the numbers in base 10. (1x1, 1x2, ..., 1x9, 2x1, 2x2, ..., 2x9, ... 9x9)
- You have more than one digit to multiply:

$$\begin{array}{r} 58 \\ \times 37 \\ \hline ? \end{array}$$

By using the operator, we can calculate:

$$7 \times 8 = 56$$

$$7 \times 5 = 35$$

$$3 \times 8 = 24$$

$$3 \times 5 = 15$$

Integer Multiplication

- How can we use these values to calculate the result?

$$\begin{array}{r} 58 \\ \times 37 \\ \hline 56 \\ 35 \\ 24 \\ + 15 \\ \hline \end{array}$$

Integer Multiplication

- We can use integer addition to find the result.

5 8	+	0 5 6	
x 3 7	+	3 5	
-----		4 0 6	← Sum 1
5 6	+	2 4	
3 5	+	0 6 4 6	← Sum 2
2 4	+	1 5	
+ 1 5	+	2 1 4 6	← Sum 3

?			

- This operation is equivalent to 16-bit multiplication using 8-bit multiplication and 8-bit addition.
- Note that the number of digits in the result is equal to the sum of the number of input digits.

Integer Multiplication

- Now assume that the digits are in base $2^8 = 256$.
(8-bit are necessary for each digit)
- Then, 16-bit multiplication is done by using 8-bit multiplication and 8-bit addition. Each 8-bit register can hold only one digit!

```
      r1 r2
    x  r3 r4
    -----
      r5 r6
      r7 r8
      r9 r10
+ r11 r12
-----
r13 r14 r15 r16
```

In fact, we do not need 16 registers to accomplish 16-bit multiplication.

If we compute the partial sums, we can re-use the registers which hold the values that are unnecessary.

Integer Multiplication

- What about negative numbers?
- If we use 2's complement format and fix the number of bits, the multiplication will give correct results for multiplication.
- 2's complement format behaves such that the negative numbers are forced to be in the positive range of a modulo of 2^n .
- For example $n = 8$, the modulo $M = 256$.


Then $-10 \pmod{256} = 246 \pmod{256}$ is also equal to 2's complement of 10.

$$(a \ b \ c \ d)_{2^8} \pmod{2^{16}} = (c \ d)_{2^8}$$

- $A \pmod{M} + B \pmod{M} = (A+B) \pmod{M}$
- $A \pmod{M} * B \pmod{M} = (A*B) \pmod{M}$
- Therefore, we compute 16-bits for 16-bit addition/multiplication.
(Not the whole 32-bits)

Integer Multiplication

- If we multiply two 16-bit numbers, we get 32-bit number. (16+16)
- We have 32-bit integers in C. On the contrary, if we multiply two integers, we again obtain 32-bit integer.
- Do we need to multiply all of the digits?
- We can omit high order digits and compute only the low 16-bit part.

$ \begin{array}{r} r1\ r2 \\ \times\ r3\ r4 \\ \hline r5\ r6 \\ r7\ r8 \\ r9\ r10 \\ +\ r11\ r12 \\ \hline r13\ r14\ r15\ r16 \end{array} $		$ \begin{array}{r} r1\ r2 \\ \times\ r3\ r4 \\ \hline r5\ r6 \\ r7\ r8 \\ +\ r9\ r10 \\ \hline r11\ r12 \end{array} $	<p>$r1 \times r3$ is not necessary.</p> <p>$r7$ and $r9$ are not used.</p>
---	--	--	---

Integer Multiplication

- Let's consider the partial sums and re-use free registers.

```
      r1 r2
x     r3 r4
-----
      r5 r6 ← r4 x r2
+  r7 r8 ← r4 x r1
-----
      r5
+  r7 r8 ← r3 x r2
-----
      r5 r6
```

Further optimizations can be done depending on the CPU architecture.

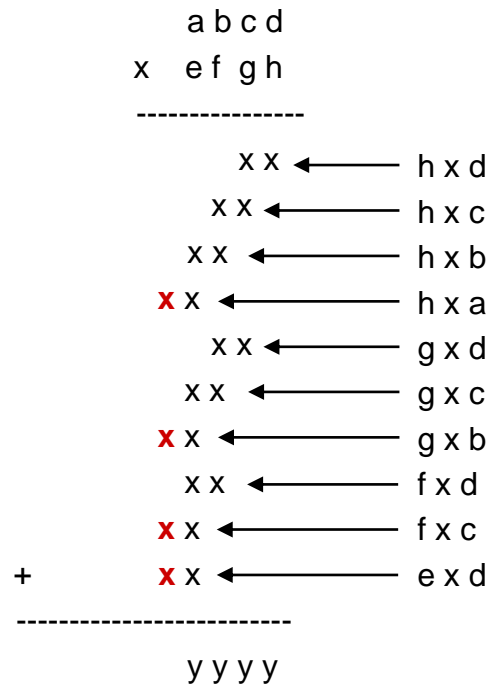
-Register limitations?

-Number of registers?

-Allowed registers for addition and multiplication?

Integer Multiplication

- What about 32-bit multiplication?
- We need 4 registers for each number.



Try to optimize 32-bit multiplication by computing partial sums.

Fixed-Point Numbers

- Fixed-point numbers are generally stored in “In.Qm” format (sometimes referred as Qn.m format)
- n = number of bits in integer part.
- m = number of bits in fractional part.
- Example: 18.Q16

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}
0	0	1	0	1	1	1	0	.	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

$$= 32 + 8 + 4 + 2 + 1/2 + 1/4 + 1/16$$

$$= 46.8125$$

Signed Fixed-Point Numbers

- Positive fixed-point numbers are the same as unsigned fixed-point numbers.
- Negative fixed-point numbers are obtained by simply calculating 2's complement as they are integers.

I8.Q8: 01000110.1100000 = 70.75

2's comp. 10111001.0100000 = -70.75

Fixed-Point Addition

- Fixed-point addition is the same as integer addition!
- Align two fixed point number and apply integer addition:

$$\begin{array}{r} r1\ r2 . r3\ r4 \\ +\ r5\ r6 . r7\ r8 \\ \hline \end{array}$$

_____ . _____

Unsigned Fixed-Point Multiplication

- Unsigned fixed-point multiplication is similar to integer multiplication.
- Consider the following multiplications:

$$\begin{array}{r} 58 \\ \times 37 \\ \hline 2146 \end{array}$$

$$\begin{array}{r} 5.8 \\ \times 3.7 \\ \hline 21.46 \end{array} \quad \begin{array}{ll} I1.Q1 & Ia.Qb \\ I1.Q1 & Ic.Qd \\ I2.Q2 & I(a+c).Q(b+d) \end{array}$$

Just multiply like integer multiplication. Align the numbers according to the point (.)

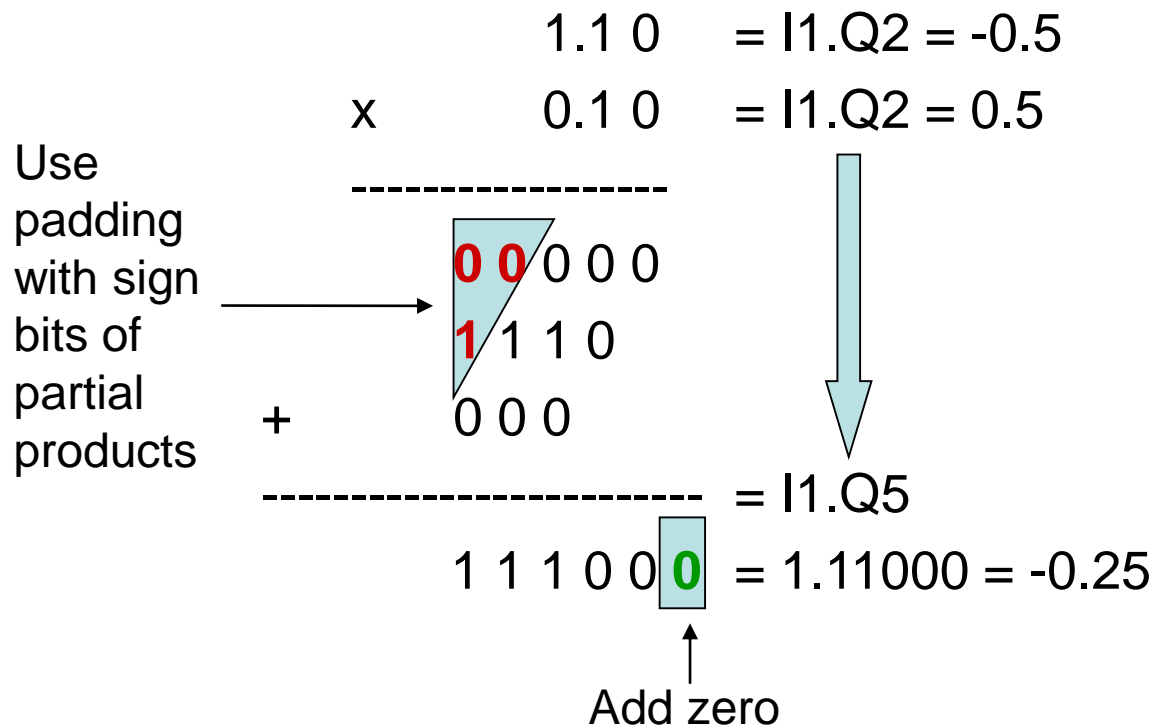
Unsigned Fixed-Point Multiplication

$$\begin{array}{r} \\ x r1.r2 \\ r3.r4 \\ \hline r6 \longleftarrow r4xr2 \\ \longleftarrow r4xr1 \\ \longleftarrow r3xr2 \\ + r12 \longleftarrow r3xr1 \\ \hline r13 r16 \end{array}$$

You can optimize the operation by considering the partial sums and the output format you need (Im.Qn).

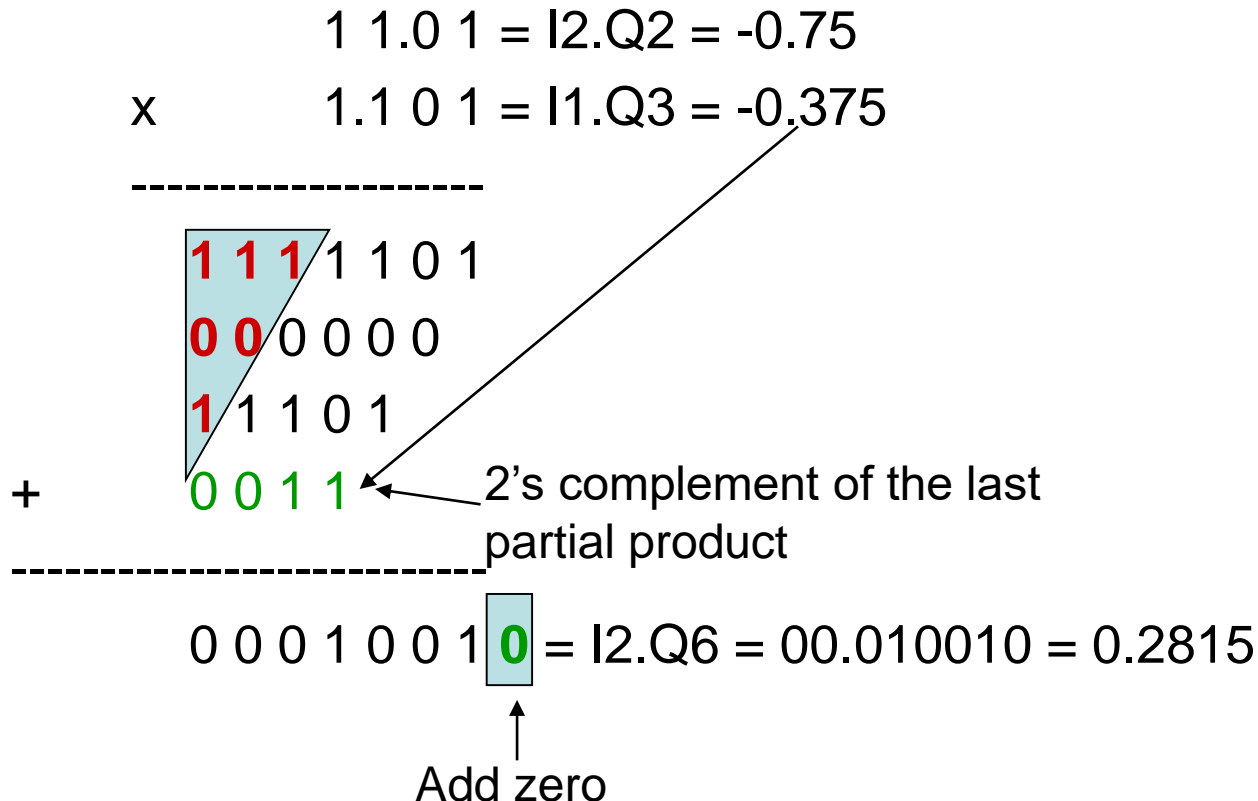
Signed Fixed-Point Multiplication

- Use 2's complement format for fixed-point numbers.
- $(Ia.Qb) * (Ic.Qd) = I(a+c-1) . Q(b+d+1)$
- Take 2's complement of the last partial product if multiplier is negative!



Signed Fixed-Point Multiplication

- Example:



Signed Fixed-Point Multiplication

- How can we use registers (e.g. 8-bit) to accomplish 16-bit (or more) signed fixed-point multiplication?
- Alternative solution 1:
 - Take 2's complement of negative numbers.
 - Apply unsigned fixed-point multiplication.
 - Finally, Take 2's complement of the result if necessary.
- Alternative solution 2:
 - 16-bit signed fixed-point multiplication is equivalent to 32-bit unsigned fixed-point multiplication (hence similar to 32-bit integer multiplication).

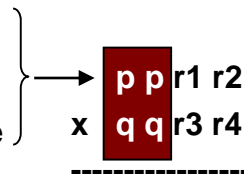
Signed Fixed-Point Multiplication

- 16-bit signed fixed-point multiplication (I8.Q8):

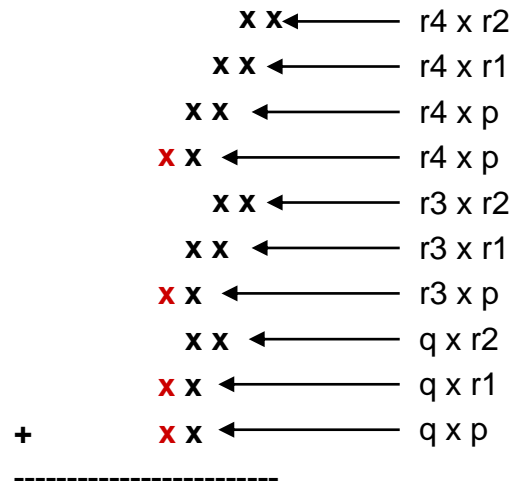
Use padding:

All zeros if the number is positive

All ones if the number is negative



x r1.r2
 r3.r4



y y.y y → Output is in (I16.Q16)