Exercise 3

2013-10-18/560 Points

Pls. mail your homework (converted into PDF File) to coa.2012.assignment@gmail.com.

File name: ID_YourName_HW_Sequence.pdf, ex: **5XXXXXXXXX_Obama_HW_01.pdf**（You could save the doc/docx as PDF in word 2007 or later, using ‘ Save as … ’ ）

Exercise 3.2

Hexadecimal (base 16) is also a commonly used numbering system for representing values in computers. In fact, it has become much more popular than octal. The following table shows pairs of hexadecimal numbers.

	A	B
a.	0D34	DD17
b.	BA1D	3617

3.2.1 [5] <3.2> What is the sum of A and B if they represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

3.2.2 [5] <3.2> What is the sum of A and B if they represent signed 16-bit hexadecimal numbers stored in sign-magnitude format? The result should be written in hexadecimal. Show your work.

3.2.3 [10] <3.2> Convert A into a decimal number, assuming it is unsigned. Repeat assuming it stored in sign-magnitude format. Show your work.

The following table also shows pairs of hexadecimal numbers.

	A	B
a.	BA7C	241A
b.	AADF	47BE

3.2.4 [5] <3.2> What is A – B if they represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

3.2.5 [5] <3.2> What is A – B if they represent signed 16-bit hexadecimal numbers stored in sign-magnitude format? The result should be written in hexadecimal. Show your work.

3.2.6 [10] <3.2> Convert A into a binary number. What makes base 16 (hexadecimal) an attractive numbering system for representing values in computers?

Exercise 3.3

Overflow occurs when a result is too large to be represented accurately given a finite word size. Underflow occurs when a number is too small to be represented correctly—a negative result when doing unsigned arithmetic, for example. (The case when a positive result is generated by the addition of two negative integers is also referred to as underflow by many, but in this textbook, that is considered an overflow). The following table shows pairs of decimal numbers.

	A	B
a.	69	90
b.	102	44

3.3.1 [5] <3.2> Assume A and B are unsigned 8-bit decimal integers. Calculate A – B. Is there overflow, underflow, or neither?

3.3.2 [5] <3.2> Assume A and B are signed 8-bit decimal integers stored in sign-magnitude format. Calculate A + B. Is there overflow, underflow, or neither?

3.3.3 [5] <3.2> Assume A and B are signed 8-bit decimal integers stored in sign-magnitude format. Calculate A – B. Is there overflow, underflow, or neither?

The following table also shows pairs of decimal numbers.

	A	B
a.	200	103
b.	247	237

3.3.4 [10] <3.2> Assume A and B are signed 8-bit decimal integers stored in two's-complement format. Calculate A + B using saturating arithmetic. The result should be written in decimal. Show your work.

3.3.5 [10] <3.2> Assume A and B are signed 8-bit decimal integers stored in two's-complement format. Calculate A – B using saturating arithmetic. The result should be written in decimal. Show your work.

3.3.6 [10] <3.2> Assume A and B are unsigned 8-bit integers. Calculate A + B using saturating arithmetic. The result should be written in decimal. Show your work.

Exercise 3.6

In this exercise we will look at a couple of other ways to improve the performance of multiplication, based primarily on doing more shifts and fewer arithmetic operations. The following table shows pairs of hexadecimal numbers.

	A	B
a.	24	c9
b.	41	18

3.6.1 [20] <3.3> As discussed in the text, one possible performance enhancement is to do a shift and add instead of an actual multiplication. Since 9×6, for example, can be written $(2 \times 2 \times 2 + 1) \times 6$, we can calculate 9×6 by shifting 6 to the left three times and then adding 6 to that result. Show the best way to calculate A × B using shifts and adds/subtracts. Assume that A and B are 8-bit unsigned integers.

3.6.2 [20] <3.3> Show the best way to calculate A × B using shift and adds, if A and B are 8-bit signed integers stored in sign-magnitude format.

3.6.3 [60] <3.3> Write a MIPS assembly language program that performs a multiplication on signed integers using shift and adds, as described in 3.6.1.

The following table shows further pairs of hexadecimal numbers.

	A	B
a.	42	36
b.	9F	8E

3.6.4 [30] <3.3> Booth's algorithm is another approach to reducing the number of arithmetic operations necessary to perform a multiplication. This algorithm has been around for years, and details about how it works are available on the Web. Basically, it assumes that a shift takes less time than an add or subtract, and uses this fact to reduce the number of arithmetic operations necessary to perform a multiply. It works by identifying runs of 1s and 0s, and performing shifts during the runs. Find a description of the algorithm and explain in detail how it works.

3.6.5 [30] <3.3> Show the step-by-step result of multiplying A and B, using Booth's algorithm. Assume A and B are 8-bit two's-complement integers, stored in hexadecimal format.

3.6.6 [60] <3.3> Write a MIPS assembly language program to perform the multiplication of A and B using Booth's algorithm.

Exercise 3.10

In a Von Neumann architecture, groups of bits have no intrinsic meanings by themselves. What a bit pattern represents depends entirely on how it is used. The following table shows bit patterns expressed in hexademical notation.

a.	0x24A60004
b.	0xAFBF0000

3.10.1 [5] <3.5> What decimal number does the bit pattern represent if it is a two's-complement integer? An unsigned integer?

3.10.2 [10] <3.5> If this bit pattern is placed into the Instruction Register, what MIPS instruction will be executed?

3.10.3 [10] <3.5> What decimal number does the bit pattern represent if it is a floating-point number? Use the IEEE 754 standard.

The following table shows decimal numbers.

a.	-1609.5
b.	-938.8125

3.10.4 [10] <3.5> Write down the binary representation of the decimal number, assuming the IEEE 754 single precision format.

3.10.5 [10] <3.5> Write down the binary representation of the decimal number, assuming the IEEE 754 double precision format.

3.10.6 [10] <3.5> Write down the binary representation of the decimal number assuming it was stored using the single precision IBM format (base 16, instead of base 2, with 7 bits of exponent).

Exercise 3.11

In the IEEE 754 floating point standard the exponent is stored in "bias" (also known as "Excess-N") format. This approach was selected because we want an all-zero pattern to be as close to zero as possible. Because of the use of a hidden 1, if we were to represent the exponent in two's-complement format an all-zero pattern would actually be the number 1! (Remember, anything raised to the zeroth power is 1, so $1.0^0 = 1$.) There are many other aspects of the IEEE 754 standard that exist in order to help hardware floating-point units work more quickly. However, in many older machines floating-point calculations were handled in software, and therefore other formats were used. The following table shows decimal numbers.

a.	5.00736125×10^5
b.	$-2.691650390625 \times 10^{-2}$

3.11.1 [20] <3.5> Write down the binary bit pattern assuming a format similar to that employed by the DEC PDP-8 (leftmost 12 bits are the exponent stored as a two's-complement number, and the rightmost 24 bits are the mantissa stored as a two's-complement number.) No hidden 1 is used. Comment on how the range and accuracy of this 36-bit pattern compares to the single and double precision IEEE 754 standards.

3.11.2 [20] <3.5> NVIDIA has a "half" format, which is similar to IEEE 754 except that it is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and stored in excess-16 format, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern assuming this format. Comment on how the range and accuracy of this 16-bit pattern compares to the single precision IEEE 754 standard.

3.11.3 [20] <3.5> The Hewlett-Packard 2114, 2115, and 2116 used a format with the leftmost 16 bits being the mantissa stored in two's-complement format, followed by another 16-bit field which had the leftmost 8 bits an extension of the mantissa (making the mantissa 24 bits long), and the rightmost 8 bits representing the exponent. However, in an interesting twist, the exponent was stored in sign-magnitude format with the sign bit on the far right! Write down the bit pattern assuming this format. No hidden 1 is used. Comment on how the range and accuracy of this 32-bit pattern compares to the single precision IEEE 754 standard.

The following table shows pairs of decimal numbers.

	A	B
a.	-1278×10^3	-3.90625×10^{-1}
b.	2.3109375×10^1	$6.391601562 \times 10^{-1}$

3.11.4 [20] <3.5> Calculate the sum of A and B by hand, assuming A and B are stored in the 16-bit NVIDIA format described in Exercise 3.11.2 (and also described in the text). Assume one guard, one round bit and one sticky bit, and round to the nearest even. Show all the steps.

3.11.5 [60] <3.5> Write a MIPS assembly language program to calculate the sum of A and B, assuming they are stored in the 16-bit NVIDIA format described in Exercise 3.11.2 (and also described in the text). Assume one guard, one round and one sticky bit, and round to the nearest even.

3.11.6 [60] <3.5> Write a MIPS assembly language program to calculate the sum of A and B, assuming they are stored using the format described in Exercise 3.11.1. Now modify the program to calculate the sum assuming the format described in Exercise 3.11.3. Which format is easier for a programmer to deal with? How do they each compare to the IEEE 754 format? (Do not worry about sticky bits for this question.)