Kaleb Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Homework 4
Computer Architecture
CSCI 361, Fall 2014
Due: November 3, 2014 in class

Problem 1 [20 pts]

Using a table similar to that shown in Figure 3.6, calculate the product of the hexadecimal unsigned 8-bit integers 62 and 12 using the hardware described in Figure 3.5. You should show the contents of each register on each step.

```
62 in hex : 0x3e → binary: 0011 1110
12 in hex : 0x0c → binary: 1100
```

operator will be 12 bits to represent all possible values of Mplier and Mcand.

Step	Mplier	Mcand	Result
Initial	1100	0000 0011 1110	0000 0000 0000
1: zero, no op	110**0**	0000 0011 1110	0000 0000 0000
2: Shift left Mcand	1100	**0000 0111 1100**	0000 0000 0000
3: Shift right Mplier	**0110**	0000 0111 1100	0000 0000 0000
1: zero, no op	011**0**	0000 0111 1100	0000 0000 0000
2: Shift left Mcand	0110	**0000 1111 1000**	0000 0000 0000
3: Shift right Mplier	**0011**	0000 1111 1000	0000 0000 0000
1: 1=>prod=prod+Mcand	001**1**	0000 1111 1000	0000 1111 1000
2: Shift left Mcand	0011	**0001 1111 0000**	0000 1111 1000
3: Shift right Mplier	**0001**	0001 1111 0000	0000 1111 1000
1: 1=>prod=prod+Mcand	000**1**	0001 1111 0000	**0010 1110 1000**
2: Shift left Mcand	0001	**0011 1110 0000**	0010 1110 1000
3: Shift right Mplier	**0000**	0011 1110 0000	0010 1110 1000
Done!			

Final Result = 0010 1110 1000 = 512 + 128 + 64 + 32 + 8 = 744 = 62 * 12 = 744!

Kaleb Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 2 [10 pts]

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 × 2 × 2 + 1) × 6, we can calculate 9 × 6 by shifting 6 to the left 3 times and then adding 6 to that result.

Show the best way to calculate 0x33 × 0x55 using shifts and adds/subtracts. Assume both inputs are 8-bit unsigned integers.

0x33 = (3*16) + 3 = 51 = 0011 0011 = 2^5 + 2^4 + 2^1 + 1

0x55 = (5*16) + 5 = 85 = 0101 0101

```
                                    1 1111 111  (Ones on this line are carry bits)
1: shift 55 left 5 places (because of 2^5)  = 1010 1010 0000
2: shift 55 left 4 places (2^4)             = 0101 0101 0000
3: shift 55 left 1 places (2^1)             = 0000 1010 1010
4: shift 55 left 0 places                   = 0000 0101 0101
                                            ----------------
add (1) + (2) + (3) + (4)                   1 0000 1110 1111
```

1	0	0	0	0	1	1	1	0	1	1	1	1
4096	2048	1024	512	256	128	64	32	16	8	4	2	1

result 4096 + 128 + 64 + 32 + 8 + 4 + 2 + 1 = **4335**
51 * 85 = 4335 IT WORKED!

Kaleb Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 3 [20 pts]
Using a table similar to that shown in Figure 3.10, calculate 74 divided by 21 using the hardware described in Figure 3.8. You should show the contents of each register on each step. Assume both inputs are unsigned 6-bit integers.

74 = 0100 1010   74/21 = 3 with a remainder of 74 – 63 = 11
21 = 0001 0101

Step	Quot	Div	Rem
Initial	0000	0001 0101	0000 1011
1: Rem = Rem - Div	0000	0001 0101	**1000 1010 (-10)**
2: Rem < 0,+Div,sll Q,Q0=0	**0000**	0001 0101	0000 1011
3: Shift Div right	0000	**0000 1010**	0000 1011
1: Rem = Rem - Div	0000	0000 1010	**0000 0001**
2: Rem >= 0, sll Q, Q0 = 1	**0001**	0000 1010	0000 0001
3: Shift Div right	0001	**0000 0101**	0000 0001
1: Rem = Rem - Div	0001	0000 0101	**1000 0100 (-4)**
2: Rem >= 0, sll Q, Q0 = 1	**0010**	0000 0101	0000 0001
3: Shift Div right	0010	**0000 0010**	0000 0001
1: Rem = Rem - Div	0010	0000 0010	**1000 0001 (-1)**
2: Rem < 0,+Div,sll Q,Q0=0	**0100**	0000 0010	0000 0001
3: Shift Div right	0100	**0000 0001**	0000 0001
1: Rem = Rem - Div	0100	0000 0001	**0000 0000**
DONE			

Final Result = 3 with a remainder of 11

Kaleb Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 4 [20 pts]

Consider the value:

`0x0C00 0000`

(a) Convert the above hex number to a bit pattern.

0	C	0	0	0	0	0	0
0000	1100	0000	0000	0000	0000	0000	0000

0000 1100 0000 0000 0000 0000 0000 0000

(b) What decimal number does the above bit pattern represent if it is an unsigned integer?

0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
				134217728	67108864	33554432	16777216	8388608	4194304	2097152	1048576	524288	262144	131072	65536	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

134,217,728 + 67,108,864 = **201,326,592**

(c) What decimal number does the above bit pattern if it is a two's complement integer?

201,326,592 no change because the most significant bit is a zero.

(d) If the above bit pattern is placed into the Instruction Register, which MIPS instruction will be executed?

This would execute a jump and link **"jal"** because the opcode is 3 binary or 0x03 hex

(e) What decimal number does the above bit pattern represent if it is a floating point number? Use the IEEE 754 standard.

|s | e | f |

0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
+ or -	128	64	32	16	8	4	2	1	.5	.25	.125	.0625	.03125	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Sign bit: `0`

Exponent is: `0001 1000 (24) - 127 = -103`

Fraction: `000...000`

(-1)^s * (1 + f) * 2^e = -1^0 * 1 * 2^-103 = **`9.8607613e-32`** so in other words:

`0.000...00098607613 or in binary: 0.0...(100 zeros)...01` `(102 zeros total and a 1 after that).`

Kaleb Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 5 [10 pts]

(a) Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 single precision format.

	Before	After
Step 1: convert to binary	63.25	**0011 1111.01**
Step 2: Normalize	0011 1111.01	**001.1111 1010 x 2^5**
Step 3: Convert the exponent to excess-127 notation	5	5 + 127 = **132**
Step 4: Convert the exponent to 8-bit binary notation	132	**1000 0100**
Step 5: Convert the fraction to "hidden bit" format.	1.1111 1010	**1111 1010**

Step 6: Identify:

Sign: = 0
Exponent: = 1000 0100
Fraction = 1111 1010 0000 0000 0000 000

Sign	Exponent (8 bits)	Fraction (23 bits)
0	**10000100**	**11111010000000000000000**

(b) Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 double precision format.

Same as above except the following: Exponent = 11 bits, Fraction = 52 bits

Sign	Exponent (11 bits)	Fraction (52 bits)
0	**00010000100**	**1111101000000000000000000000000000000000000000000000**

Problem 6 [10 pts]

(a) Using the IEEE 754 floating point format single precision, write down the bit pattern that would represent −1/4. Can you represent −1/4 exactly?

-1/4 = -0.25 = 0.01 (binary) = 1.000 X 2^-2 (normalized): -2 + 127 = 125 = 0111 1101

Sign	Exponent (8 bits)	Fraction (23 bits)
1	**01111101**	**00000000000000000000000**

Yes you can represent -1/4 exactly

(b) What do you get if you add −1/4 to itself 4 times? What is −1/4 × 4? Are they the same? What should they be?

-1/4 + -1/4 + -1/4 + -1/4 = -4/4 = -1

-1/4(4) = -1 **YES They are the same!**

What they should be: **Nothing different, they are correct.**

Kaleb Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

(c) Using the IEEE 754 floating point format single precision, find the bit pattern that would represent 1/3. Can it be represented exactly? If not, round upwards.
Right off the bat we know this is impossible as 1/3 exactly is:
0.3333333333333333333333333333333333333333333333333333333333............to infinity and beyond.
Therefore the closest we can get is whatever number is 0.3333333333...2? or something similar so as n approaches 1/3 without going over...
target: 0.33333333....
power: (-2)
result: 0

Power	check	Result
-1 (1/2)	0.5 | > t	No good
-2 (1/4)	**0.25** < t	Ok
-3 (1/8)	.125 + .25 = .375	No good
-4 (1/16)	.0625 + .25 = **.3125**	Ok
-5 (1/32)	.3125 + .03125 = .34375	No good
-6 (1/64)	.3125 + .015625 = **.328125**	Ok
-7 (1/128)	.328125 + .0078125 = .3359375	No good
-8 (1/256)	.328125 + .00390625 = **.33203125**	Ok
-9 (1/512)	.33203125 + .001953125 = .333984375	No good
-10 (1/1024)	.33203125 + .0009765625 = **.3330078125**	Ok
-11 2048	Pattern emerging, skip every other, check at end.	Assume No Good
-12 4096	Result: **.333251953**	Ok
-14 16384	Result: **.3333129883**	Ok
-16 65536	Result: **.3333282471**	Ok
-18 262144	Result: .3333320618	Ok
-20 1048576	Result: .3333330154	Ok
-22 4194304	Result: .3333332539	Ok
-23 8388608	Check our assumption: Result: .3333333731	No good

Final result would be the binary .0101 0101 0101 0101 0101 010
normalize: 1.01 0101 0101 0101 0101 010 X 2^-2
Exponent: -2 + 127 = 125 = 0111 1101
Fraction: 0101 0101 0101 0101 0101 010

Sign	Exponent (8 bits)	Fraction (23 bits)
0	**01111101**	**01010101010101010101010**

That is the closest we can get to .33333333333...to infinity and beyond... if we were using double, we would set every even bit to 1 and every odd bit to zero in the fraction out to 52.

Kaleb Himes
kaleb@wolfssl.com
kaleb.himes@gmail.com

Problem 7 [10 pts]

IEEE 754-2008 contains a half precision that it is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed.
Write down the bit pattern to represent -1.5625×10^{-1} assuming a version of this format, which uses an excess-16 format to store the exponent.
Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

	Before	After
Step 1: convert to binary	-0.15625	**0.00101**
Step 2: Normalize	0.00101	**1.01 X 2^-3**
Step 3: Convert the exponent to excess-127 notation	-3	**-3 + 15 = 12**
Step 4: Convert the exponent to 5-bit binary notation	12	**0 1100**
Step 5: Convert the fraction to "hidden bit" format.	1.01	**01**

Step 6: Identify:

Sign: = 1
Exponent: = 0 1100
Fraction = 01 0000 0000

Sign	Exponent (5 bits)	Fraction (10 bits)
1	**01100**	**0100000000**

Our limitations here on decimal values would be between 2^-1 and 2^-14 vs the 2^-23 precision we can achieve with single precision.
Integer values would be limited to between 0 and 2048 exactly, anything larger would be rounded to multiples of 2, 4, 8, 16, and 32.