Ryan Young

CSE 3666

HW 4

3.12) [20] <§3.3> Using a table similar to that shown in Figure 3.6, calculate the product of the octal unsigned 6-bit integers 62 and 12 using the hardware described in Figure 3.3. You should show the contents of each register on each step.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Iteration Number | Step | Multiplier # | Multiplicand # | Product # |
| 0 | Starting Values | 001 010 | 000 000  110 010 | 000 000 000  000 |
| 1 | 1. 0, no operation done 2. shift left on the multiplicand 3. shift right the multiplier | 1. 001 010 2. 001 010 3. 000 101 | 1. 000 000   110 010   1. 000 001   100 100   1. 000 001   100 100 | 1. 000 000 000   000   1. 000 000 000   000   1. 000 000 000   000 |
| 2 | 1. 1, product = product + multiplicand 2. shift left the multiplicand 3. shift right the multiplier | 1. 000 101 2. 000 101 3. 000 010 | 1. 000 001   100 100   1. 000 011   001 000   1. 000 011   001 000 | 1. 000 001 100   100   1. 000 001 100   100   1. 000 001 100   100 |
| 3 | 1. 0, No operation 2. Shift left the multiplicand 3. Shift right the multiplier | 1. 000 010 2. 000 010 3. 000 001 | 1. 000 011   001 000   1. 000 110   010 000   1. 000 110   010 000 | 1. 000 001 100   100   1. 000 001 100   100   1. 000 001 100   100 |
| 4 | 1. 1, product = product + multiplicand 2. Shift left the multiplicand 3. Shift right the multiplier | 1. 000 001 2. 000 001 3. 000 000 | 1. 000 110   010 000   1. 001 100   100 000   1. 001 100   100 000 | 1. 000 111 100   100   1. 000 111 100   100   1. 000 111 100   100 |

Result: 000 111 110 100 = 7648

3.18) [20] <§3.4> 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.

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Quotient | Divisor | Remainder |
| Initial Values | 0000 | 1010 1000 | 0100 1010 |
| 1 Remainder = Remainder – divisor  SLL Quotient  SRL Divisor | 0000 | 0101 0100 | 1010 0010 |
| 2 Remainder = Remainder – divisor  SLL Quotient  SRL Divisor | 0000 | 0010 1010 | 0100 1010 |
| 3 Remainder = Remainder – divisor  SLL Quotient  SRL Divisor | 0001 | 0001 0101 | 0010 0000 |
| 4 Remainder = Remainder – divisor  SLL Quotient  SRL Divisor | 0011 | 0000 1010 | 0000 1011 |

Giving us an answer of 00112 = 310 with a remainder of 0000 10112 = 1110.

3.20) [5]<§3.5> What decimal number does the bit pattern 0x0c000000 represent if it is a two’s complement integer, unsigned integer?

Converting this number to binary: 0000 1010 0000 0000 0000 0000 0000 00002. So, since the leading bit is a zero, we know this number is positive. This binary number would thus represent 12 \* 166 = 20132659210 for both two’s complement and as an unsigned integer because it is a positive number.

3.21) If the bit pattern 0x0C000000 is placed into the Instruction Register, what MIPS instruction will be executed?

Converting the hex number to binary we get 0000 1100 0000 0000 0000 0000 0000 00002. The first six digits give us the op code which is 0000 112 or 310. This means that this function is the Jump and Link function and since the target is 0 it will jump there and make the $ra register take the address of the pc register + 4.

3.22) What decimal number does the bit pattern 0x0C000000 represent if it is a floating-point number?

Use the IEEE 754 standard.

Converting the hexadecimal number to binary first we get 0000 1100 0000 0000 0000 0000 0000 00002.

Since the first bit is 0 we know that the number is positive, the next 8 bits give us the exponent: 0001 1000 and the mantissa is given by the remaining 23 bits 0000 0000 0000 0000 000. This yields a floating-point value of 9.860761315262648 \* 10-32.

3.23) What decimal number does the bit pattern 0x0C000000 represent if it is a floating-point number?

Use the IEEE 754 single precision format.

63.25 is a positive number so the first bit, the sign bit is going to be 0. Converting 63 to binary is 63 / 2 = 1

31 / 2 = 1

15/2 = 1

7/2 = 1

3 / 2 = 1

0.5 / 2 = 1

6310 = 1111112

0.2510­ converted to binary is:

0.25 \* 2 = 0

0.5 \* 2 = 1

= 102

Converting 63.25 to binary is 0100 0010 0111 1101 0000 0000 0000 00002

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

63.25 is a positive number so the first bit, the sign bit is going to be 0. Converting 63 to binary is 63 / 2 = 1

31 / 2 = 1

15/2 = 1

7/2 = 1

3 / 2 = 1

0.5 / 2 = 1

6310 = 1111112

0.2510­ converted to binary is:

0.25 \* 2 = 0

0.5 \* 2 = 1

= 102

In order to normalize 63.25 we would need to more the decimal point 5 places to the left, since the bias for IEEE 754 Double Precision is 1023 the binary exponent used will be 1028. 1028 represented in binary is 100000001002. Combining everything together this yields 0100 0000 1001 1111 0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 00002.

3.27) IEEE 754-2008 contains a half precision that is only 16 bits wide. The left most 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 \* 101 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.

The first bit is going to be a 1 because the number is a negative number. 1 in decimal is 1 in binary so that will be the same and the decimal portion:

0.15625 \* 2 = 0

0.3125 \* 2 = 0

0.625 \* 2 = 1

0.25 \* 2 = 0

0.5 \* 2 = 1

Reading bottom to top: 101002.

The exponent is found by subtracting 1510 – 310 = 1210 = 011002.

The mantissa is now: 1 01100 0100 0000 00 0100 0000 002.

So, finally for a 754-2008 representation of the number we get 1 01100 0100 0000 002.

3.29) Calculate the sum of 2.6125 \* 101 and 4.150390625 \* 101 by hand, assuming A and B are stored in the 16-bit half precision described in Exercise 3.27. Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps.

26.125 = x 23 = = 2= 11010.0012 = 1.1010001x242

0.4150390625 = x210 = = 2 = 0.01101010012 = 1.10101001x2-22

1.1010001000 0000

+ .0000011010 1001

Normalize scientific notation as well:

1.1010100010 1001 \* 24

Rounding with one round bit results in:

1.1010100010\*24

= 0.5 + 0.125 + 0.03125 + 0.001953125 = 0.658203125

1 \* (1 + 0.658204) \*219-15 = 26.531264

3.41) Using the IEE 754 floating point format, write down the bit pattern that would represent -1/4. Can you represent -1/4 exactly?

Yes, you can represent -1/4 exactly, the number is negative so the sign bit will be 1. Converting ¼ to binary: ¼ \* 2 = 0

½ \* 2 = 1

So ¼ = 10

In Single Precision IEEE 754 the bias is 127 so in order to get the desired -2 exponent the binary exponent will have to represent 125. Thus, making the value: 1011 1111 1110 1000 0000 0000 0000 00002. For Double Precision IEEE 754 the bias is 1023 so to get the desired -2 exponent the binary exponent will have to represent 1021. Thus, making the value 1011 1111 1101 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 00002.