ECE 485/585 – Computer Organization and Design

HOMEWORK #3 SOLUTION

1. Provided numbers: $6FD4_{16}$ and $273B_{16}$
   $6FD4_{16}$ = 0110 1111 1101 0100 = +28628
   $273B_{16}$ = 0010 0111 0011 1011 = +10043
   $6FD4_{16} – 273B_{16} = 6FD4_{16} + (-273B_{16})$
   2's comp. of 273B = 1101 1000 1100 0100 = -10043
   0110 1111 1101 0100
   <u>+ 1101 1000 1100 0101</u>
   1 0100 1000 1001 1001 = $+4899_{16}$ (pos. and neg. added together => ignore overflow)
   Answer: 4899_{16}

2. Provided numbers: 174_{10} and 85_{10}
   174_{10} = 1010 1110 = -46
   85_{10} = 0101 0101 = +85
   2's comp of 46 = 1101 0010
   1101 00
   <u>+ 0101 0101</u>
   1 0010 0111 = +39
   Answer: +39, neither overflow nor underflow (ignored due to adding pos. and neg. number)

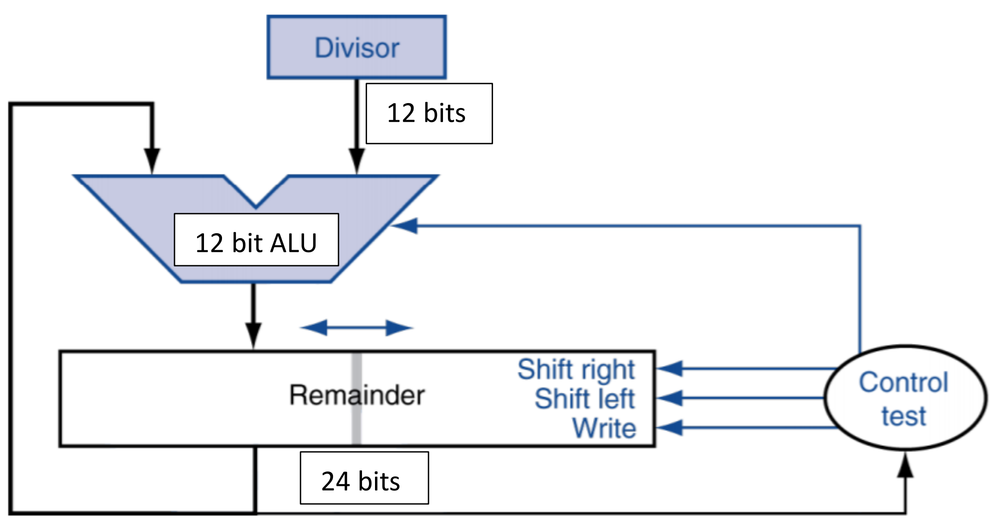
3. See the table on the next page

ECE 485/585 – Computer Organization and Design

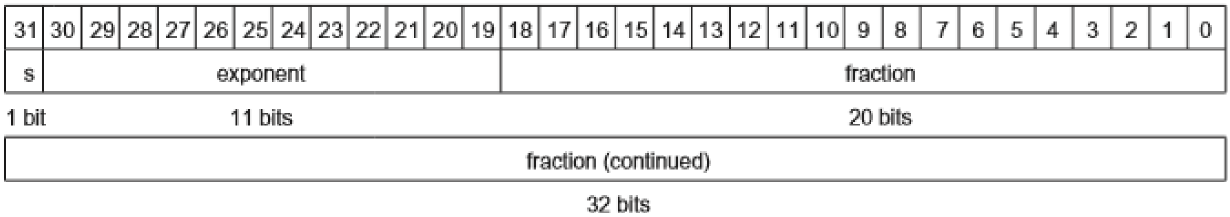
Iteration	Step	Quotient	Divisor	Remainder
0	Initial Values	000 000	010 110 000 000	000 000 111 100
1	1: Rem = Rem - Div	000 000	010 110 000 000	101 010 111 100
	2b: Rem < 0 => +Div, sll Q, Q0=0	000 000	010 110 000 000	000 000 111 100
	3: Shift Div right	000 000	001 011 000 000	000 000 111 100
2	1: Rem = Rem - Div	000 000	001 011 000 000	110 101 111 100
	2b: Rem < 0 => +Div, sll Q, Q0=0	000 000	001 011 000 000	000 000 111 100
	3: Shift Div right	000 000	000 101 100 000	000 000 111 100
3	1: Rem = Rem - Div	000 000	000 101 100 000	111 011 011 100
	2b: Rem < 0 => +Div, sll Q, Q0=0	000 000	000 101 100 000	000 000 111 100
	3: Shift Div right	000 000	000 010 110 000	000 000 111 100
4	1: Rem = Rem - Div	000 000	000 010 110 000	111 110 001 100
	2b: Rem < 0 => +Div, sll Q, Q0=0	000 000	000 010 110 000	000 000 111 100
	3: Shift Div right	000 000	000 001 011 000	000 000 111 100
5	1: Rem = Rem - Div	000 000	000 001 011 000	111 111 100 100
	2b: Rem < 0 => +Div, sll Q, Q0=0	000 000	000 001 011 000	000 000 111 100
	3: Shift Div right	000 000	000 000 101 100	000 000 111 100
6	1: Rem = Rem - Div	000 000	000 000 101 100	000 000 010 000
	2a: Rem > 0 => sll Q, Q0 =1	000 001	000 000 101 100	000 000 010 000
	3: Shift Div right	000 001	000 000 010 110	000 000 010 000
7	1: Rem = Rem - Div	000 001	000 000 010 110	111 111 101 010
	2b: Rem < 0 => +Div, sll Q, Q0=0	000 010	000 000 010 110	000 000 010 000
	3: Shift Div right	000 010	000 000 001 011	000 000 010 000

Result: Quotient = 2, Remainder = 16. 2*22 = 44 + 16 = 60 as we expect

The figure below shows the changes to our divisor design: We need to decrease the divisor and ALU from 32 bits to 12 bits. Also, we drop the remainder from 64 bits to 24 bits.



4. Write down the binary representation of the double precision floating point number 56.93. Double precision floating points are of the following form:
(please note that this figure from the text has an extra bit for the exponent, which should stop at bit 20)



First we need to convert 56.93 into binary
$56 = 111000_2$
For the fraction:

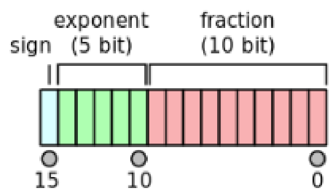
0.93*2 = 1.86 Integral Part = 1
0.86*2 = 1.72 Integral Part = 1
0.72*2 = 1.44 Integral Part = 1
0.44*2 = 0.88 Integral Part = 0
0.88*2 = 1.76 Integral Part = 1
0.76*2 = 1.52 Integral Part = 1
0.52*2 = 1.04 Integral Part = 1
0.04*2 = 0.08 Integral Part = 0
0.08*2 = 0.16 Integral Part = 0
0.16*2 = 0.32 Integral Part = 0
0.32*2 = 0.64 Integral Part = 0
0.64*2 = 1.28 Integral Part = 1
0.28*2 = 0.56 Integral Part = 0
0.56*2 = 1.12 Integral Part = 1
0.12*2 = 0.24 Integral Part = 0
0.24*2 = 0.48 Integral Part = 0
0.48*2 = 0.96 Integral Part = 0
0.96*2 = 1.92 Integral Part = 1
0.92*2 = 1.84 Integral Part = 1
0.84*2 = 1.68 Integral Part = 1
0.68*2 = 1.36 Integral Part = 1
0.36*2 = 0.72 Integral Part = 0
0.72*2 = 1.44 Integral Part = 1

**Found same number as iteration 3, so will repeat endlessly
So in total: 56.93 = 111000.1110111000010100011110101110000101000111101…
First we shift our number so that we have a leading 1 and then the decimal place:
1.11000111011100001010001111010… * 2^5
We have a sign bit of 0 and now need to calculate the exponent = 5 + 1023 = 1028
$1028 = 100\ 0000\ 0100_2$
We have a sign bit of 0 in this case.

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>S</td><td colspan="11">exponent</td><td colspan="20">Fraction</td></tr>
<tr><td>0</td><td colspan="11">100 0000 0100</td><td colspan="20">1100 0111 0111 0000 1010</td></tr>
<tr><td colspan="32">Fraction (continued)</td></tr>
<tr><td colspan="32">0011 1101 0111 0000 1010 0011 1101 0111</td></tr>
</table>

After filling up our space, we truncate the remaining bits

5. Write down the half-precision representation of the following number: -1.585 * 10^{-1}. Half Precision floating point numbers are of the following form:



First we need to convert 0.1585 into binary

0.1585*2 = 0.317 Integral Part = 0
0.317 *2 = 0.634 Integral Part = 0
0.634 *2 = 1.268 Integral Part = 1
0.268 *2 = 0.536 Integral Part = 0
0.536 *2 = 1.072 Integral Part = 1
0.072 *2 = 0.144 Integral Part = 0
0.144 *2 = 0.288 Integral Part = 0
0.288 *2 = 0.576 Integral Part = 0
0.576 *2 = 1.152 Integral Part = 1
0.152 *2 = 0.304 Integral Part = 0
0.304 *2 = 0.608 Integral Part = 0
0.608 *2 = 1.216 Integral Part = 1
0.216 *2 = 0.432 Integral Part = 0

Truncate the rest (this is enough values for our floating point number)
This results in 0.0010100010010 = 1.0100010010 * 2^{-3}
We have a sign bit of 1 and now need to calculate the exponent = -3 + 15 = 12
12 = 01100_2
So we get the following floating point number:

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>S</td><td colspan="5">exponent</td><td colspan="10">Fraction</td></tr>
<tr><td>1</td><td colspan="5">011 00</td><td colspan="10">01 0001 0010</td></tr>
</table>

When compared to single-precision floating point, we receive a better range for the number of bits that we use (5/16 compared to 8/32) however we sacrifice precision in doing so.