Homework Assignment - 3

Submitted By
Fahd Humayun
168000889 (fh186)

Question – 1:

A and B are 8-bit two's complement integers.

$$A = C4_{16} = 1100\ 0100_2 = -60_{10}$$

$$-A = 0011\ 1100_2 = 60_{10}$$

$$B = 4D_{16} = 0100\ 1101_2 = 77_{10}$$

$$A * B = EDF4_{16} = 1110\ 1101\ 1111\ 0100 = -4620_{10}$$

Step	Multiplicand (8-bits)	Product Register (16-bits)	Current Bit	Previous Bit	Next Operation
0	*1100 0100*	*0000 0000 0100 1101*	*1*	*0 (mythical)*	*Subtract*
1a.	*1100 0100*	*0011 1100 0100 1101*	*1*	*0*	*Shift*
1b.	*1100 0100*	*0001 1110 0010 0110*	*0*	*1*	*Add*
2a.	*1100 0100*	*1110 0010 0010 0110*	*0*	*1*	*Shift*
2b.	*1100 0100*	*1111 0001 0001 0011*	*1*	*0*	*Subtract*
3a.	*1100 0100*	*0010 1101 0001 0011*	*1*	*0*	*Shift*
3b.	*1100 0100*	*0001 0110 1000 1001*	*1*	*1*	*Shift*
4.	*1100 0100*	*0000 1011 0100 0100*	*0*	*1*	*Add*
5a.	*1100 0100*	*1100 1111 0100 0100*	*0*	*1*	*Shift*
5b.	*1100 0100*	*1110 0111 1010 0010*	*0*	*0*	*Shift*
6.	*1100 0100*	*1111 0011 1101 0001*	*1*	*0*	*Subtract*
7a.	*1100 0100*	*0010 1111 1101 0001*	*1*	*0*	*Shift*
7b.	*1100 0100*	*0001 0111 1110 1000*	*0*	*1*	*Add*
8a.	*1100 0100*	*1101 1011 1110 1000*	*0*	*1*	*Shift*
8b.	***1100 0100***	***1110 1101 1111 0100***	***x***	***x***	***DONE***

10 - Subtract in next operation is done by adding two's compliment of multiplicand to the left half of the product and then storing result in the left of product.

01 - Add in next operation is done by just adding the multiplicand directly to the left half of the product and then storing result in the left of product.

00-11-(and after sub/add) Shift in next operation is done by shifting the product to right 1 bit and sign extending.

Question – 2 (a):

A and B are 8-bit unsigned number.

$$A = C4_{16} = 1100\ 0100_2 = 196_{10}$$

$$B = 4D_{16} = 0100\ 1101_2 = 77_{10}$$

$$A * B = 3AF4_{16} = 0011\ 1010\ 1111\ 0100 = 15092_{10}$$

Step	Multiplicand (16-bits)	Product Register (16-bits)	Multiplier (8-bits)	LSB of Multiplier	Next Operation
0	*0000 0000 1100 0100*	*0000 0000 0000 0000*	*0100 1101*	*1*	*Add*
1a.	*0000 0000 1100 0100*	*0000 0000 1100 0100*	*0100 1101*	*1*	*Shift left multiplicand*
1b.	*0000 0001 1000 1000*	*0000 0000 1100 0100*	*0100 1101*	*1*	*Shift right multiplier*
1c.	*0000 0001 1000 1000*	*0000 0000 1100 0100*	*0010 0110*	*0*	*Shift left multiplicand*
2a.	*0000 0011 0001 0000*	*0000 0000 1100 0100*	*0010 0110*	*0*	*Shift right multiplier*
2b.	*0000 0011 0001 0000*	*0000 0000 1100 0100*	*0001 0011*	*1*	*Add*
3a.	*0000 0011 0001 0000*	*0000 0011 1101 0100*	*0001 0011*	*1*	*Shift left multiplicand*
3b.	*0000 0110 0010 0000*	*0000 0011 1101 0100*	*0001 0011*	*1*	*Shift right multiplier*
3c.	*0000 0110 0010 0000*	*0000 0011 1101 0100*	*0000 1001*	*1*	*Add*
4a.	*0000 0110 0010 0000*	*0000 1001 1111 0100*	*0000 1001*	*1*	*Shift left multiplicand*
4b.	*0000 1100 0100 0000*	*0000 1001 1111 0100*	*0000 1001*	*1*	*Shift right multiplier*
4c.	*0000 1100 0100 0000*	*0000 1001 1111 0100*	*0000 0100*	*0*	*Shift left multiplicand*
5a.	*0001 1000 1000 0000*	*0000 1001 1111 0100*	*0000 0100*	*0*	*Shift right multiplier*
5b.	*0001 1000 1000 0000*	*0000 1001 1111 0100*	*0000 0010*	*0*	*Shift left multiplicand*
6a.	*0011 0001 0000 0000*	*0000 1001 1111 0100*	*0000 0010*	*0*	*Shift right multiplier*
6b.	*0011 0001 0000 0000*	*0000 1001 1111 0100*	*0000 0001*	*1*	*Add*
7a.	*0011 0001 0000 0000*	*0011 1010 1111 0100*	*0000 0001*	*1*	*Shift left multiplicand*
7b.	*0110 0010 0000 0000*	*0011 1010 1111 0100*	*0000 0001*	*1*	*Shift right multiplier*
7c.	*0110 0010 0000 0000*	*0011 1010 1111 0100*	*0000 0000*	*0*	*Shift left multiplicand*
8a.	*1100 0100 0000 0000*	*0011 1010 1111 0100*	*0000 0000*	*0*	*Shift right multiplier*
8b.	***1100 0100 0000 0000***	***0011 1010 1111 0100***	***0000 0000***	***0***	***DONE***

If LSB of multiplier is 1 then add multiplicand to product, shift multiplicand left 1 bit, and shift multiplier right 1 bit.

If LSB of multiplier is 0 then shift multiplicand left 1 bit, and shift multiplier right 1 bit.

Question – 2 (b):

Each repetition in worst case has:

$$1\ addition\ operation\ (2ns)\ \&\ 2\ shift\ operations\ (2ns\ each)$$

Where, the 2 shift operations can be done simultaneously, therefore,

$$1\ addition\ operation\ +\ 2\ shift\ operations\ =\ 2ns\ +\ 2ns\ =\ 4ns/repetition$$

So, each repetition of worst case takes $4ns$. For 8-bit numbers there will be 8 repetitions therefore,

$$\boldsymbol{the\ worst\ case\ time = 8 * 4ns = 32\ ns}$$

Question – 3:

```
# switch(s){
# case0:    h = i + 3j; break;
# case1:    h = j - 4i; break;
# }

# $s0 = s, $s1, = i, $s2 = j, $s3 = h, sequential words in memory starting at $s4

# switch (s)
    sll     $t0, $s0, 2     # t0 = 4*s
    add     $t1, $t0, $s4   # t1 = starting address + 4*s (i.e. address of switch(s))
    lw      $t2, 0($t1)     # load word from memory (i.e. switch(s))
    jr      $t2             # jump to address loaded from memory stored in $t2

case0:
    sll     $t3, $s2, 1     # t3 = 2j
    add     $t3, $t3, $s2   # t3 = 2j + j = 3j
    add     $s3, $s1, $t3   # s3 = h = i + 3j
    j       EXIT            # jump to EXIT

case1:
    sll     $t4, $s1, 2     # t4 = 4i
    sub     $s3, $s2, $t4   # s3 = h = j - 4i
    j       EXIT            # jump to EXIT

EXIT:
    ...
```

Question – 4 (a):

A & *B* 4-bit unsigned numbers.

$$A = 0111_2 = 7_{10}$$

$$B = 0101_2 = 5_{10}$$

$$\frac{A}{B} \rightarrow Remainder = 0010_2 = 2_{10}$$

$$\frac{A}{B} \rightarrow Quotient = 0001_2 = 1_{10}$$

Dividend is loaded into the remainder register. Shift left the remainder 1 bit, then do iterations with the beginning step of subtracting the divisor register from the left half of the remainder register and place the result in the left half of the remainder register. If the remainder is < 0 (MSB of remainder = 1) then restore the original value by adding the divisor to the left half of the remainder, and then shift the remainder register to left with setting the new LSB of remainder to 0. If the remainder >= 0 (MSB of remainder = 0) then shift the remainder to left with setting the new LSB of remainder to 1. After the iterations shift the left half of remainder to right 1 bit.

Quotient (N/A)	Divisor (4-bits)	Remainder (8-bits)	Description	Step	Time (ns)
0111	*0101*	*0000 0111*	*Initial values*	*0*	
1110	*0101*	*0000 1110*	*Shift left 1 bit*	*0a.*	*2*
1110	*0101*	***1**011 1110*	*Rem = Rem - Div*	*1a.*	*2*
1110	*0101*	*0000 1110*	*Rem = Rem + Div*	*1b.*	*2*
1100	*0101*	*0001 1100*	*Shift left Rem, LSB = 0*	*1c.*	*2*
1100	*0101*	***1**100 1100*	*Rem = Rem - Div*	*2a.*	*2*
1100	*0101*	*0001 1100*	*Rem = Rem + Div*	*2b.*	*2*
1000	*0101*	*0011 1000*	*Shift left Rem, LSB = 0*	*2c.*	*2*
1000	*0101*	***1**110 1000*	*Rem = Rem - Div*	*3a.*	*2*
1000	*0101*	*0011 1000*	*Rem = Rem + Div*	*3b.*	*2*
0000	*0101*	*0111 0000*	*Shift left Rem, LSB = 0*	*3c.*	*2*
0000	*0101*	***0**010 0000*	*Rem = Rem - Div*	*4a.*	*2*
0001	*0101*	*0100 0001*	*Shift left Rem, LSB = 1*	*4b.*	*2*
0001	*0101*	***0010 0001***	*Shift right (left half) Rem*	*Done*	*2*
		Remainder Quotient			*Total: 26 ns*

The quotient column in this algorithm is not needed (not applicable), but the values in the quotient reflects the values in the right half of the remainder.

Question – 4 (b):

Before the repetitions/iterations start there is a $shift$ operation in the beginning that takes $2ns$.

In the iteration process there are two possible paths that will follow:

1. One of the path is for remainder greater than 0 after the $subtraction$ operation that takes $2ns$, which is followed by a $shift$ operation that also takes $2ns$. So, in total this path of the iteration takes $4ns$.
2. The second path is for remainder less than 0 after the $subtraction$ operation that takes $2ns$, which is followed by an $addition$ operation that takes $2ns$, followed by a $shift$ operation that takes $2ns$. So, in total this path of the iteration takes $6ns$.

At the end of the iterations there is a final $shift$ operation of $2ns$.

So, for the above division, the total time necessary to perform using the given hardware is calculated in the table above i.e. $\mathbf{26ns}$.

Question – 5 (a) & (b):

Reg	Sign (1-bit)	Exponent (8-bits)	Fraction (23-bits)	Decimal equivalent
$f1	*1*	*0101 0100*	*1101 0101 0011 0100 0000 011*	$\mathbf{-2.084 * 10^{-13}}$
$f2	*1*	*0101 0100*	*1001 0110 1000 0100 0001 010*	$\mathbf{-1.805 * 10^{-13}}$
$f3	*0*	*0100 0010*	*0110 1110 1010 1110 1100 001*	$\mathbf{6.212 * 10^{-19}}$

The equation to convert from binary floating point representation to decimal is

$$(number)_{10} = (-1)^{sign} * (1 + fraction) * 2^{power\,=\,exponent-bias}$$

For $\$f1$,

$sign = 1$
$exponent = 2^6 + 2^4 + 2^2 = 64 + 16 + 4 = 84$
$bias = 127\ (single\ precision)$
$power = 84 - 127 = -43$
$fraction_2 = 0.1101\ 0101\ 0011\ 0100\ 0000\ 011$
$1 + fraction_2 = 1.1101\ 0101\ 0011\ 0100\ 0000\ 011$
$\mathbf{\$f1 = (-1)^1 * (1.1101\ 0101\ 0011\ 0100\ 0000\ 011) * 2^{-43}}$
$fraction_{10} = 2^{-1} + 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + 2^{-11} + 2^{-12} + 2^{-14} + 2^{-22} + 2^{-23} = 0.833$
$1 + fraction_{10} = 1.833$
$(number)_{10} = (-1)^1 * (1.833) * 2^{-43} = \mathbf{-2.084 * 10^{-13}}$

Similarly for $\$f2$ & $\$f3$,

For $\$f2$,

$sign = 1$
$exponent = 84$
$bias = 127\ (single\ precision)$
$power = 84 - 127 = -43$
$fraction_2 = 0.1001\ 0110\ 1000\ 0100\ 0001\ 010$
$1 + fraction_2 = 1.1001\ 0110\ 1000\ 0100\ 0001\ 010$
$\mathbf{\$f2 = (-1)^1 * (1.1001\ 0110\ 1000\ 0100\ 0001\ 010) * 2^{-43}}$
$fraction_{10} = 0.588$
$1 + fraction_{10} = 1.588$
$(number)_{10} = \mathbf{-1.805 * 10^{-13}}$

For $\$f3$,

$sign = 1$
$exponent = 66$
$bias = 127\ (single\ precision)$
$power = 66 - 127 = -61$
$fraction_2 = 0.0110\ 1110\ 1010\ 1110\ 1100\ 001$
$1 + fraction_2 = 1.0110\ 1110\ 1010\ 1110\ 1100\ 001$
$\mathbf{\$f3 = (-1)^0 * (1.0110\ 1110\ 1010\ 1110\ 1100\ 001) * 2^{-61}}$
$fraction_{10} = 0.432$
$1 + fraction_{10} = 1.432$
$(number)_{10} = \mathbf{6.212 * 10^{-19}}$

Addition:

$\$f1 + \$f2$:

Step # 1: Align binary points (shift number with smaller exponent)

This step is not needed as the numbers are already aligned.

Step # 2: Add significands

$= (-1.11010101001101000000011) * 2^{-43} + (-1.10010110100001000001010) * 2^{-43}$
$= -11.01101011101110000001101 * 2^{-43}$

Step # 3: Normalize result & check for over/underflow

$\$f1 + \$f2 = -1.10110101110111000001101 * 2^{-42}$

Step # 4: Round and renormalize (not needed)

Addition:

$(\$f1 + \$f2) + \$f3$:

Step # 1: Align binary points (shift number with smaller exponent)

Shifting $f3

$\$f3 = 0.00000000000000000010110 * 2^{-42}$

Step # 2: Add significands

Two's compliment of $\$f1 + \$f2 = 0.01001010001000111110010$

$= 0.01001010001000111111001 * 2^{-42} + 0.00000000000000000010110 * 2^{-42}$
$= 0.01001010001001000001111 * 2^{-42}$

Two's compliment of $(\$\boldsymbol{f1} + \$\boldsymbol{f2}) + \$\boldsymbol{f3} = \boldsymbol{-1.10110101110110111110000 * 2^{-42}}$

Step # 3: Normalize result & check for over/underflow (not needed)

Step # 4: Round and renormalize (not needed)

$$sign = 1$$
$$exponent = power + bias = -42 + 127 = 85_{10} = 01010101_2$$
$$fraction = .1011\ 0101\ 1101\ 1011\ 1110\ 000$$
$$1 + fraction_{10} = 1.710$$
$$(number)_{10} = -1.710 * 2^{-42} = -3.889 * 10^{-13}$$

So, the final result,

Sign (1-bit)	Exponent (8-bits)	Fraction (23-bits)	Decimal equivalent
1	*0101 0101*	*1011 0101 1101 1011 1110 000*	$\boldsymbol{-3.889 * 10^{-13}}$

Question – 6 (a) & (b):

Multiplication:

$\$f1 * \$f2$
$= -1.11010101001101000000011 * 2^{-43} * -1.10010110100001000001010 * 2^{-43}$

Step # 1: Add exponents

$unbiased = -43 + (-43) = -86$

Step # 2: Multiply significands

$= 1.11010101001101000000011 * 1.10010110100001000001010$
$= 10.11101001000100101001010 * 2^{-86}$

Step # 3: Normalize result & check for over/underflow

$\$f1 * \$f2 = 1.01110100100010010100 1010 * 2^{-85}$

$exponent = -85 + 127 = 42$ *(no overflow/underflow)*

Step # 4: Round and renormalize (not needed)

Step # 5: Determine sign:

$negative * negative = positive$

$\$f1 * \$f2 = 1.01110100100010010100 1010 * 2^{-85}$

Multiplication:

$(\$f1 * \$f2) * \$f3$
$= 1.01110100100001001010010 * 2^{-85} * 1.01101110101011101100001 * 2^{-61}$

Step # 1: Add exponents

$unbiased = -85 + (-61) = -146$

Step # 2: Multiply significands

$= 1.01110100100001001010010 * 1.01101110101011101100001$
$= 10.00010101100110101001011 * 2^{-146}$

Step # 3: Normalize result & check for over/underflow

$(\$f1 * \$f2) * \$f3 = 1.00001010110011010100101 1 * 2^{-145}$

$exponent = -145 + 127 = -18$

Which means result is underflow and ($f1*$f2)$f3 cannot be calculated*
