COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE
14:332:331
Rutgers University
Spring 2016

Homework 3 (Solution)

1. Suppose A = C4 and B = 4D (both in hexadecimal). Show the step by step result multiplying A and B, using Booth's algorithm. Assume A and B are 8-bit two's complement integers, stored in hexadecimal format.

Solution:

A = C4 = $1100\ 0100_2 = -60_{10}$
B = 4D = $0100\ 1101_2 = 77_{10}$
-A = -C4 = $0011\ 1100_2 = 60_{10}$

Step	Product Register	Multiplicand	Current Bit	Previous Bit	Operation
1a.	0000 0000 0100 1101	1100 0100	1	0	Subtract
1b.	0011 1100 0100 1101	1100 0100	1	0	shift right
2a.	0001 1110 0010 0110	1100 0100	0	1	Add
2b.	1110 0010 0010 0110	1100 0100	0	1	shift right
3a.	1111 0001 0001 0011	1100 0100	1	0	Subtract
3b.	0010 1101 0001 0011	1100 0100	1	0	shift right
4.	0001 0110 1000 1001	1100 0100	1	1	nothing/shift right
5a.	0000 1011 0100 0100	1100 0100	0	1	Add
5b.	1100 1111 0100 0100	1100 0100	0	1	shift right
6.	1110 0111 1010 0010	1100 0100	0	0	nothing/shift right
7a.	1111 0011 1101 0001	1100 0100	1	0	Subtract
7b.	0010 1111 1101 0001	1100 0100	1	0	shift right
8a	0001 0111 1110 1000	1100 0100	0	1	Add
8b	1101 1011 1110 1000	1100 0100	0	1	shift right
Result	1110 1101 1111 0100	-	-	-	-

Result = 1110 1101 1111 0100_2 = 0xEDF4 = -4620_{10}

2. Suppose A = C4 and B = 4D (both in hexadecimal). Show the step by step result multiplying A and B, using the multiplier hardware shown in Fig. 1. Assume A and B are 8-bit unsigned numbers, stored in hexadecimal format.

   b) Suppose for an 8-bit number, each step of operation (either addition or shift) takes 2ns. Please calculate the time necessary to perform a multiply using the approach given in Fig. 1. Assume the registers have been initialized. In hardware, please note that the shifts of the multiplicand and multiplier can be done simultaneously.

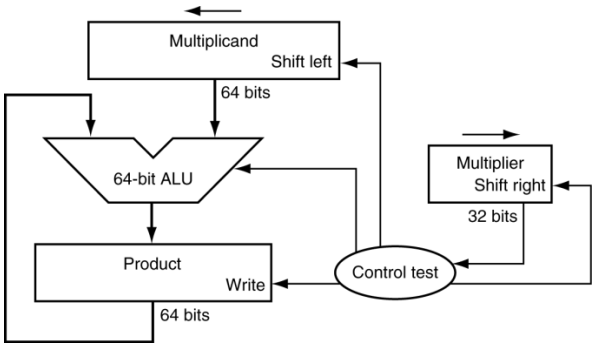

Fig. 1

Solution:

A = C4 = 1100 0100_2 = 196_{10}
B = 4D = 0100 1101_2 = 77_{10}

Step	Product Register	Multiplier	Multiplicand	Multiplier LSB	Operation
1a.	0000 0000 0000 0000	0100 1101	0000 0000 1100 0100	1	add
1b.	0000 0000 1100 0100	0100 1101	0000 0000 1100 0100	1	shift left multiplicand
1c.	0000 0000 1100 0100	0100 1101	0000 0001 1000 1000	1	shift right multiplier
2a.	0000 0000 1100 0100	0010 0110	0000 0001 1000 1000	0	shift left multiplicand
2b.	0000 0000 1100 0100	0010 0110	0000 0011 0001 0000	0	shift right multiplier
3a.	0000 0000 1100 0100	0001 0011	0000 0011 0001 0000	1	add
3b.	0000 0011 1101 0100	0001 0011	0000 0011 0001 0000	1	shift left multiplicand
3c.	0000 0011 1101 0100	0001 0011	0000 0110 0010 0000	1	shift right multiplier
4a.	0000 0011 1101 0100	0000 1001	0000 0110 0010 0000	1	add

4b.	0000 1001 1111 0100	0000 1001	0000 0110 0010 0000	1	shift left multiplicand
4c.	0000 1001 1111 0100	0000 1001	0000 1100 0100 0000	1	shift right multiplier
5a.	0000 1001 1111 0100	0000 0100	0000 1100 0100 0000	0	shift left multiplicand
5b.	0000 1001 1111 0100	0000 0100	0001 1000 1000 0000	0	shift right multiplier
6a.	0000 1001 1111 0100	0000 0010	0001 1000 1000 0000	0	shift left multiplicand
6b.	0000 1001 1111 0100	0000 0010	0011 0001 0000 0000	0	shift right multiplier
7a.	0000 1001 1111 0100	0000 0001	0011 0001 0000 0000	1	add
7b.	0011 1010 1111 0100	0000 0001	0011 0001 0000 0000	1	shift left multiplicand
7c.	0011 1010 1111 0100	0000 0001	0110 0010 0000 0000	1	shift right multiplier
8a.	0011 1010 1111 0100	0000 0000	0110 0010 0000 0000	0	shift left multiplicand
8b.	0011 1010 1111 0100	0000 0000	1100 0100 0000 0000	0	shift right multiplier
Result	**0011 1010 1111 0100**	-	-	-	-

Result: $0011\ 1010\ 1111\ 0100_2 = 0x3AF4 = 15092_{10}$

b. 8 bits * (1 addition operation/bit + 1 shift operation/bit) * 2 ns/operation
= 8 bits * (2 operations/bit) * 2 ns/operation = **32 ns**

3. Compile the assembly code for the following C code (using jr). Assume that s, i, j, and h are in $s0,$s1, and $s2, and $s3, respectively and that sequential words in memory starting at $s4.

```
switch(s){
    case 0: h=i+3j; break;
    case 1: h=j-4i; break;
}
```

Solution:

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

```
sll $s5, $s0, 2          #$s5 = 4*s
add $s5, $s5, $s4              #$s5 = addr of switch[k]
lw $s6, 0($s5)           #$s6 = switch[k]
jr $s6                         #jump to address in $s6

Case0:
        add $s7, $s2, $s2      # s7 = 2*j
        add $s7, $s7, $s2      # s7 = 3*j
        add $s3, $s1, $s7      #h = i + 3j
        j exit                 #jump to exit label
Case1:
        sll $s8, $s1, 2        #$s8 = 4*i
        sub $s3, $s2, $s8      #h = j-4i
        j exit                 #jump to exit label

exit:
        ...
```

4. Suppose A = 0111 and B = 0101. Show the step by step result computing A divide B, using the hardware shown in Fig. 2. Assume A and B are 4-bit unsigned numbers. Dividend is initially loaded into the Remainder register. Please show the steps in the given table. Please expand the table if more rows needed.

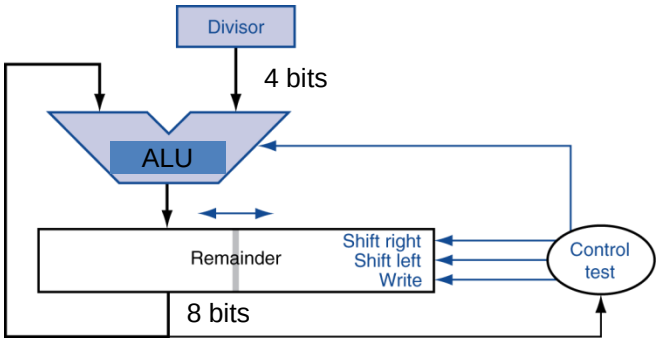

Fig. 2

b) Suppose that each operation (addition, subtraction or shift) takes 2ns. Please calculate the time necessary to perform the above division using the given hardware. Assume the registers have been initialized.

Solution:

$A = 0111_2$
$B = 0101_2$

$-B = 1011_2$

The Remainder register holds both the quotient and the remainder. The quotient will be formed in the 4 most significant bits of the remainder register.

Step	Divisor	Remainder	Description
0.	0101	0000 0111	Initial Value
1a.	0101	0000 1110	shift left remainder
1b.	0101	1011 1110	subtract divisor
1c.	0101	0000 1110	add back divisor
2a.	0101	0001 1100	shift left remainder
2b.	0101	1100 1100	subtract divisor
2c.	0101	0001 1100	add back divisor
3a.	0101	0011 1000	shift left remainder
3b.	0101	1110 1000	subtract divisor
3c.	0101	0011 1000	add back divisor
4a.	0101	0111 0000	shift left remainder
4b.	0101	0010 0000	subtract divisor
4c.	0101	0100 0001	shift left remainder and set the new rightmost bit to 1
result	0101	0010 0001	Shift left half of remainder to right 1
			Done

Result: So the result is 0010 0001, where 0010 is the remainder and 0001 is the quotient.
b. 13 operations * 2 ns / operation = **26 ns**

5. Assume that IEEE 754 floating point format is used and $f1 and $f2 and $f3 store the floating numbers as below:

$f1: 1010 1010 0110 1010 1001 1010 0000 0011
$f2: 1010 1010 0100 1011 0100 0010 0000 1010
$f3: 0010 0001 0011 0111 0101 0111 0110 0001

a) Please show the step by step result computing for ($f1+$f2)+$f3. (first show the steps required to compute $f1+$f2 and then the steps for adding this results to $f3)

b) Please show the values of each register in decimal as well as the value of the final result ($f1+$f2+$f3) in decimal.

Solution:

$f1:

Sign	Exponent	Fraction
1	0101 0100	110 1010 1001 1010 0000 0011

S = 1
Exponent = $0101\ 0100_2 = 84_{10}$
Fraction = $.110\ 1010\ 1001\ 1010\ 0000\ 0011_2 = 0.832825064659118652343 75_{10}$

$f1=$(-1)^1$ x 1.110 1010 1001 1010 0000 0011 x 2^{84-127}= -1.110 1010 1001 1010 0000 0011 x 2^{-43}

$f2:

Sign	Exponent	Fraction
1	010 1010 0	100 1011 0100 0010 0000 1010

S = 1
Exponent = $0101\ 0100_2 = 84_{10}$
Fraction = $.100\ 1011\ 0100\ 0010\ 0000\ 1010_2 = 0.587952852249145507812 5_{10}$

$f2=$(-1)^1$ x1. 100 1011 0100 0010 0000 1010 x 2^{84-127}= - 1. 100 1011 0100 0010 0000 1010 x 2^{-43}

$f1+$f2:

Since both numbers have the same exponent there is no need for shifting any of the two.

$f1 +$f2 = -1.110 1010 1001 1010 0000 0011_2 x 2^{-43} + -1.100 1011 0100 0010 0000 1010_2 x 2^{-43}= -11.011 0101 1101 1100 0000 1101 x 2^{-43} = -1.1011 0101 1101 1100 0000 110 x 2^{-42} (renormalized)

S = 1
Exponent = -42 + 127 = $85_{10} = 0101\ 0101_2$
Fraction =$.1011\ 0101\ 1101\ 1100\ 0000\ 110_2 = 0.710388898849487304687 5_{10}$

Sign	Exponent	Fraction
1	0101 0101	1011 0101 1101 1100 0000 110

$f3:

Sign	Exponent	Fraction
0	0100 0010	011 0111 0101 0111 0110 0001

S = 0
Exponent = 0100 $0010_2 = 66_{10}$
Fraction = .011 0111 0101 0111 0110 0001_2

$\$f3 = (-1)^0 \times (1.01101110101011101100001_2) \times 2^{66-127} = 1.01101110101011101100001 \times 2^{-61}$

($f1+$f2)+$f3:

($f1+$f2)+$f3 = $-1.10110101110111000000110 \times 2^{-42} + 1.01101110101011101100001 \times 2^{-61}$

$= -1.10110101110111000000110 \times 2^{-42} + 0.00000000000000000010110 \times 2^{-42}$

$= -1.10110101110110111110000 \times 2^{-42}$ (no need to renormalize)

S = 1
Exponent = -42 + 127 = $85_{10} = 0101\ 0101_2$
Fraction = $.10110101110110111110000_2$

Sign	Exponent	Fraction
1	0101 0101	101 1010 1110 1101 1111 0000

6. Repeat Question 5.a and show the steps required to compute ($f1*$f2)*$f3.
   b) Show the result of ($f1*$f2)*$f3 in decimal.

$f1: 1 010 1010 0 110 1010 1001 1010 0000 0011
$f2: 1 010 1010 0 100 1011 0100 0010 0000 1010
$f3: 0 010 0001 0 011 0111 0101 0111 0110 0001

Solution:

$f1:

Sign	Exponent	Fraction
1	0101 0100	110 1010 1001 1010 0000 0011

S = 1

Exponent = $0101\ 0100_2 = 84_{10}$
Fraction = $.110\ 1010\ 1001\ 1010\ 0000\ 0011_2$

$f1 = $-1.110\ 1010\ 1001\ 1010\ 0000\ 0011 \times 2^{-43}$

$f2:

Sign	Exponent	Fraction
1	010 1010 0	100 1011 0100 0010 0000 1010

S = 1
Exponent = $0101\ 0100_2 = 84_{10}$
Fraction = $.100\ 1011\ 0100\ 0010\ 0000\ 1010_2$

$f2 = $- 1. 100\ 1011\ 0100\ 0010\ 0000\ 1010 \times 2^{-43}$

$f1*f2:

Sign = 1 XOR 1 = 0 (+)
Exponent (unbiased): (-43) + (-43) = -86
$1.11010101001101000000011_2 \times 1.10010110100001000001010_2 = 10.11\ 1010\ 0100\ 0100\ 1010\ 0101\ 0011\ 1111\ 1100\ 1010\ 0001\ 1110_2 \times 2^{-86}$
First we normalize the product. Thus,
$10.11\ 1010\ 0100\ 0100\ 1010\ 0101\ 0011\ 1111\ 1100\ 1010\ 0001\ 1110_2 \times 2^{-86} =$
$1.011\ 1010\ 0100\ 0100\ 1010\ 0101\ 0011\ 1111\ 1100\ 1010\ 0001\ 1110_2 \times 2^{-85}$

There is no underflow because -127< -85 < 128.

We need to round to the correct number of bits, that is 24 in total (23 of the fraction plus 1 for the hidden bit). So, the number is
$1.011\ 1010\ 0100\ 0100\ 1010\ 0101_2 \times 2^{-85}$
Check again if there is a need to normalize, and in this case we do not have to. The result is

S = 0
Exp = -85 + 127 = 42 = $0010\ 1010_2$
Frac = $011\ 1010\ 0100\ 0100\ 1010\ 0101_2$

Sign	Exponent	Fraction
0	0010 1010	011 1010 0100 0100 1010 0101

b. ($f1*$f2)*$f3

$f3:

Sign	Exponent	Fraction
0	0100 0010	011 0111 0101 0111 0110 0001

S = 0
Exponent = $0100\ 0010_2 = 66_{10}$
Fraction = $.011\ 0111\ 0101\ 0111\ 0110\ 0001_2$

$f3 = $1.01101110101011101100001 \times 2^{-61}$

($f1*$f2)*$f3:

Sign = 0 XOR 0 = 0 (+)
Exponent (unbiased): (-85) + (-61) = -146
$1.01110100100010010100101_2 \times 1.01101110101011101100001_2 = 10.00\ 0101\ 0110\ 0110\ 1010\ 0101\ 1101\ 1011\ 0001\ 0101\ 1000\ 0101_2 \times 2^{-146}$
First we normalize the product. Thus,
$10.00\ 0101\ 0110\ 0110\ 1010\ 0101\ 1101\ 1011\ 0001\ 0101\ 1000\ 0101_2 \times 2^{-146} =$
$1.000\ 0101\ 0110\ 0110\ 1010\ 0101\ 1101\ 1011\ 0001\ 0101\ 1000\ 0101_2 \times 2^{-145}$

There is underflow because -145< -127 and we throw an exception.