COMPUTER ARCHITECTURE & ASSEMBLY LANGUAGE
Rutgers University
Homework 3
Leonardo Roman

1. A. Suppose A = C5 and B = 6D (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.

$A = 1100\ 0101_2,\ B = 0110\ 1101_2,\ AxB = -6431$

M = 1100 0101

Steps	Product	Multiplicand-Q	bit	Q-1	Operation
Initial	0000 0000	0011 110**1**	**1**	0	
1	0011 1011	0011 110**1**	1	0	P=P-M
shifting	0001 1101	1011 011**0**	**0**	1	SR 1
2	1110 0010	1011 011**0**	0	1	P=P+M
shifting	1111 0001	0101 101**1**	**1**	0	SR 1
3	0010 1100	0101 101**1**	1	0	P=P-M
shifting	0001 0110	0010 110**1**	**1**	1	SR 1
4 shifting	0000 1011	0001 011**0**	**1**	1	N/O SR1
5	1101 0000	0001 011**0**	0	1	P=P+M
shifting	1110 1000	0000 101**1**	**1**	0	SR1
6	0010 0011	0000 101**1**	**1**	0	P=P-M
shifting	0001 0001	1000 010**1**	**1**	1	SR1
7 shifting	0000 1000	1100 001**0**	**1**	1	N/O SR1
8	1100 1101	1100 001**0**	**0**	1	P=P+M
shifting	**1110 0110**	**1110 0001**			SR1

Result $1110\ 0110\ 1110\ 0001_2 = 0xE6E1 = -6431$

B. What does the mythical bit in Booth's algorithm stand for? Sketch/justify why it is needed and why it works.

If we are limited to looking at just 2 bits, we can then try to match the situation in the preceding drawing, according to the value of 2 bits. By starting with the mythical with 0 there will be two initial operations, subtract or no operation.

LSB	Myth	Explanation	Example
1	0	Beginning of a run of 1's	0000 111**1 0**000
1	1	Middle of a run of 1's	0000 1**11**1 0000
0	1	End of a run of 0s	000**0 1**111 0000
0	0	Middle of a run of 0s	0**00**0 1111 0000

2. Suppose A = C5 and B = 6D (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.

$A = 1100\ 0101_2$, $B = 0110\ 1101_2$, $AxB = -6431$

M = 1100 0101

Steps	Multiplicand	Multiplier	op	Result
Initial	0000 0000 1100 0101	0110 1101	**N/A**	0000 0000 0000 0000
1	0000 0000 **1100 0101**	0011 110**1**	add	0000 0000 1100 0101
2.a shift mulnd left	0000 000**1 1000 101**0	0110 1101		0000 0000 1100 0101
2.b shift multr right	0000 000**1 1000 101**0	0011 011**0**	N/A	0000 0000 1100 0101
3.a shift mulnd left	0000 00**11 0001 01**00	0011 0110		0000 0000 1100 0101
3.b shift multr right	0000 00**11 0001 01**00	0001 101**1**	add	0000 0011 1101 1001
4.a shift mulnd left	0000 0**110 0010 1**000	0001 1011		0000 0011 1101 1001
4.b shift multr right	0000 0**110 0010 1**000	0000 1101	**add**	0000 1010 0000 0001
5.a shift mulnd left	0000 **1100 0101** 0000	0000 1101		0000 1010 0000 0001
5.b shift multr right	0000 **1100 0101** 0000	0000 0110	**N/A**	0000 1010 0000 0001
6.a shift mulnd left	000**1 1000 101**0 0000	0000 0110		0000 1010 0000 0001
6.b shift multr right	000**1 1000 101**0 0000	0000 0011	**add**	0101 0011 1110 0001

7.a shift mulnd left	00**11 0001 01**00 0000	0000 0011		0010 0010 1010 0001
7.b shift multr right	00**11 0001 01**00 0000	0000 0001	**add**	0101 0011 1110 0001
8.a shift mulnd left	0**110 0010 1**000 0000	0000 0001		**0101 0011 1110 0001**
8.b shift multr right	0**110 0010 1**000 0000	0000 0000		Stop

Result $0101\ 0011\ 1110\ 0001_2 = 0x53E1 = 21473$

b) Suppose for an 8-bit number, each step of operation (either addition or shift) takes 2ns. Please calculate the worst case 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.

For worst case would require addition at every shift, in this case the 8-bit number would be all ones(1111 1111). This means that there would be 8 additions and 8 shifts. If multiplicand and multiplier can be done simultaneously and it takes 2ns for shifting and addition, this would give a total run time of 8(2ns)+8(2ns)=32ns.

3. Compile the assembly code for the following C code (using jr). Assume that s, i, j, and h are in $s0,$s1,$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;
}
```

```
sll $s5,$s0,2          #$s5 = 4*s
add $s5,$s5,$s4        #$s5 = s + label address
lw $s6, 0($s5)         #load label information dependent on ‘s’ into $s6
jr $s6                 #go to label
L0:
add $s7,$s2,$s2        #$s7 = 2*j
add $s7,$s7,$s2        #$s7 = 3*j
add $s3,$s1,$s7        #h = i + 3j
j exit                 #break
L1:
```

```
sll $s1,$s1,2        #4i
sub $s3,$s2,$s1      #h = j - 4i
j exit               #break
```

4. Suppose A = 0111 and B = 0101. Show the step by step result computing A divide B, using the hardware 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.

0111 ÷ 0101

n+1 steps	Description	Quotient	Divisor	*Remainder −= Divisor*
0	N/A	0000	0101 0000	0000 0111
1	Rem = rem - div	0000	0101 0000	**1**011 0111
	rem<0	0000	0101 0000	0000 0111
	Shift right divisor	0000	00**10 1**000	0000 0111
2	Rem = rem - div	0000	00**10 1**000	**1**101 1111
	rem<0	0000	00**10 1**000	0000 0111
	Shift right divisor	0000	000**1 01**00	0000 0111
3	Rem = rem - div	0000	000**1 01**00	**1**111 0011
	rem<0	0000	000**1 01**00	0000 0111
	Shift right divisor	0000	0000 **101**0	0000 0111
4	Rem = rem - div	0000	0000 **101**0	**1**111 1101
	rem<0	0000	0000 **101**0	0000 0111
	Shift right divisor	0000	0000 0**101**	0000 0111
5	Rem = rem - div	0000	0000 0**101**	**0**000 0111
	rem>0	0001	0000 0**101**	0000 0111
Stop	Shift right divisor	**0001**	0000 **0010**	

$a = dq + r \rightarrow 7 = 5 * 1 + 2$

Hence $0111 = 0101 = 0101 * 0001 + 0010$

Which is the last number of the quotient times five plus the last number of the divisor.

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.

10(2ns) = 20ns

Problem 5:

a) Calculate the product of the hexadecimal unsigned 8-bit integers 75 and 23 using the hardware described in Figure 1 (refined version of the multiply). You should show the contents of each register on each step.

$0x75 = 0111\ 0101_2 = 117_{10}$

$0x23 = 0010\ 0011_2 = 35_{10}$

$117 * 35 = 4095_{10} = 0000\ 1111\ 1111\ 1111_2 = fff_{16}$

Iteration	Operation	Multiplicand	Product
0	N/A	0111 0101	0000 0000 0010 001**1**
1	**Add mult to leftmost 4-bits of product**	0111 0101	0111 0101 0010 0011
	Shift product 1-bit to the right	0111 0101	0011 1010 1000 000**1**
2	**Add mult to leftmost 4-bits of product**	0111 0101	1010 1111 1001 0001
	Shift product 1-bit to the right	0111 0101	0101 0111 1100 100**0**
3	Shift product 1-bit to the right	0111 0101	0010 1011 1110 010**0**
4	Shift product 1-bit to the right	0111 0101	0001 0101 1111 001**0**
5	Shift product 1-bit to the right	0111 0101	0000 1010 1111 100**1**
6	**Add mult to leftmost 4-bits of product**	0111 0101	0111 1111 1111 100**1**
	Shift product 1-bit to the right	0111 0101	0011 1111 1111 110**0**
7	Shift product 1-bit to the right	0111 0101	0001 1111 1111 111**0**
8	Shift product 1-bit to the right	0111 0101	**0000 1111 1111 1111**

b) Using a table similar to that shown below, calculate 75 divided by 23 using the hardware in Figure 2. The numbers are hexadecimal unsigned 8-bit integers. You should show the contents of each register on each step.

$$75_{16} \div 23_{16} = 0111\ 0101_2 \div 0010\ 0011_2 = 117 = 35 * 3 + 12$$

Iter	Divisor	Remainder	Operation
0	0010 0011	0000 0000 0111 0101	
	0010 0011	0000 0000 1110 1010	Sll 1-bit
1	0010 0011	**1**101 1101 1110 101**0**	*Rem −= div*
	0010 0011	0000 0000 1110 1010	*Rem += div*
	0010 0011	0000 0001 1101 01**00**	Sll 1-bit
2	0010 0011	**1**101 1110 1101 0100	*Rem −= div*
	0010 0011	0000 0001 1101 0100	*Rem += div*
	0010 0011	0000 0011 1010 1**000**	Sll 1-bit
3	0010 0011	**1**110 0000 1010 1000	*Rem −= div*
	0010 0011	0000 0011 1010 1000	*Rem += div*
	0010 0011	0000 0111 0101 **0000**	Sll 1-bit
4	0010 0011	**1**110 0100 0101 0000	*Rem −= div*
	0010 0011	0000 0111 0101 0000	*Rem += div*
	0010 0011	0000 1110 101**0 0000**	Sll 1-bit
5	0010 0011	**1**110 1011 1010 0000	*Rem −= div*
	0010 0011	0000 1110 1010 0000	*Rem += div*
	0010 0011	0001 1101 01**00 0000**	Sll 1-bit
6	0010 0011	**1**111 1010 0100 0000	*Rem −= div*
	0010 0011	0001 1101 0100 0000	*Rem += div*

	0010 0011	0011 1010 1**000 0000**	Sll 1-bit
7	0010 0011	**0**001 0111 1000 000	*Rem −= div*
	0010 0011	0010 1111 **0000 0001**	Sll 1-bit,R0=1
8	0010 0011	**0**000 1100 0000 0001	*Rem −= div*
	0010 0011	0001 100**0 0000 0011**	Sll 1-bit,R0=1
	sll 1-bit left half remainder	**0000 1100 0000 0011**	ANS

Quotient = $0000\ 0011_2$ = 3

Remainder = $0000\ 1100_2$ = 12