Introduction to Computer Engineering I (ECSE-221)
Assignment 5: Computer Arithmetic
Available: Nov 17, 2006
Due: Dec 1, 2006

Please submit this assignment by 5pm, Dec 1, 2006, on WebCT. Your assignment must consist of a "C" program, entitled "Assign5-Q1-<ID>.c" for Question 1, an assembly program, "Assign5-Q2-<ID>.s" for Question 2, and a Microsoft Word document, "Assign5-Q3-<ID>.doc" for Question 3. In these filenames, <ID> should be replaced by your student ID number; for example, "Assign5-Q2-609384123.s"

In addition, your circuits, timing files, and library for Question 3 must also be submitted via WebCT as per instructions on the site. You should create a folder just for this question, and keep all the circuit, timing files, and libraries you create in that folder to include in your submission.

Your "C" and assembly programs must be fully documented: marks will be deducted if your programs do not contain sufficient comments.

This assignment will be marked out of 100 points.

Assignments received up to 24 hours late will be penalized by 10%; assignments received up to 48 hours late will be penalized by 20%, and assignments received more than 48 hours late will not be marked.

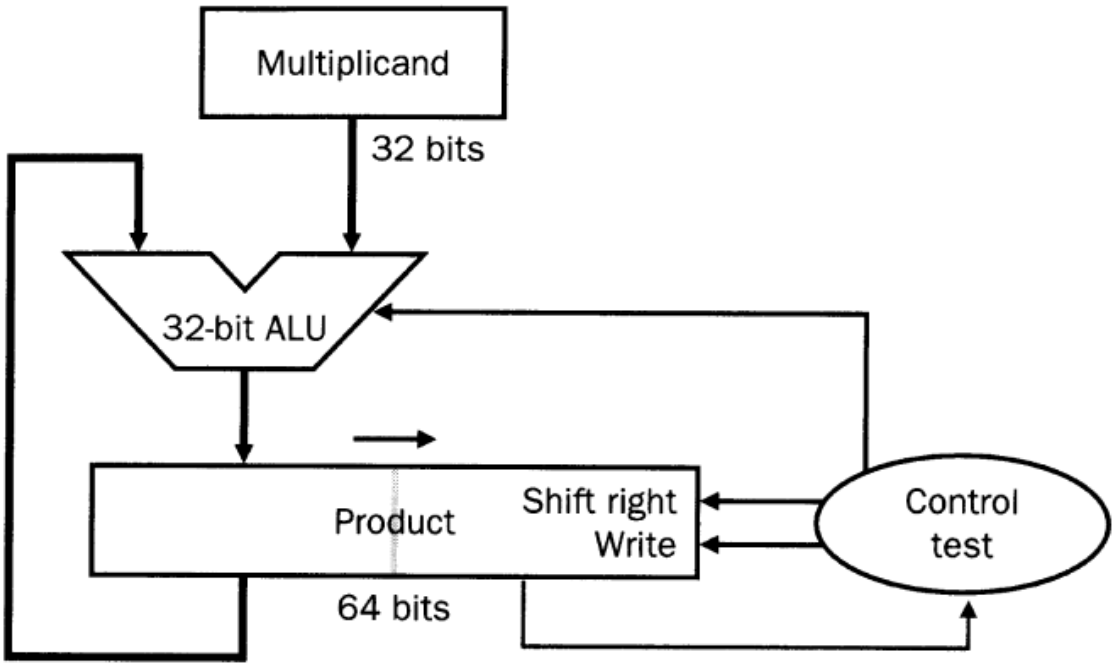
Question 1 (25 points)

Implement a “C” language function to implement *unsigned* binary multiplication, using the add-shift algorithm demonstrated in class, according to the following prototype:

```
void mult32(
     unsigned long multiplicand,
     unsigned long multiplier,
     unsigned long *productH,
     unsigned long *productL);
```

Notice that the product returned will be 64 bits, with the upper 32 bits in productH and the lower 32 bits in productL. We could have passed a 64-bit argument using an array (pointer), but this was avoided to keep the assignment to a reasonable length.

Your function must be implemented according to the following datapath:



Validate your function by writing a main() program which tests your function with different arguments taken from a table of test cases (similar to Assignment 4). It should print the multiplicand, multiplier, and then print the resulting product as two numbers: the upper 32 bits and the lower 32 bits of the product. Your program does not need to verify that the results of the multiplications are correct, but you should check the output of your program by hand to be sure that it works correctly.

Choose your test cases carefully to cover the full range of possibilities, including multiplications using one, zero, and the largest possible numbers.

Question 2 (25 points)

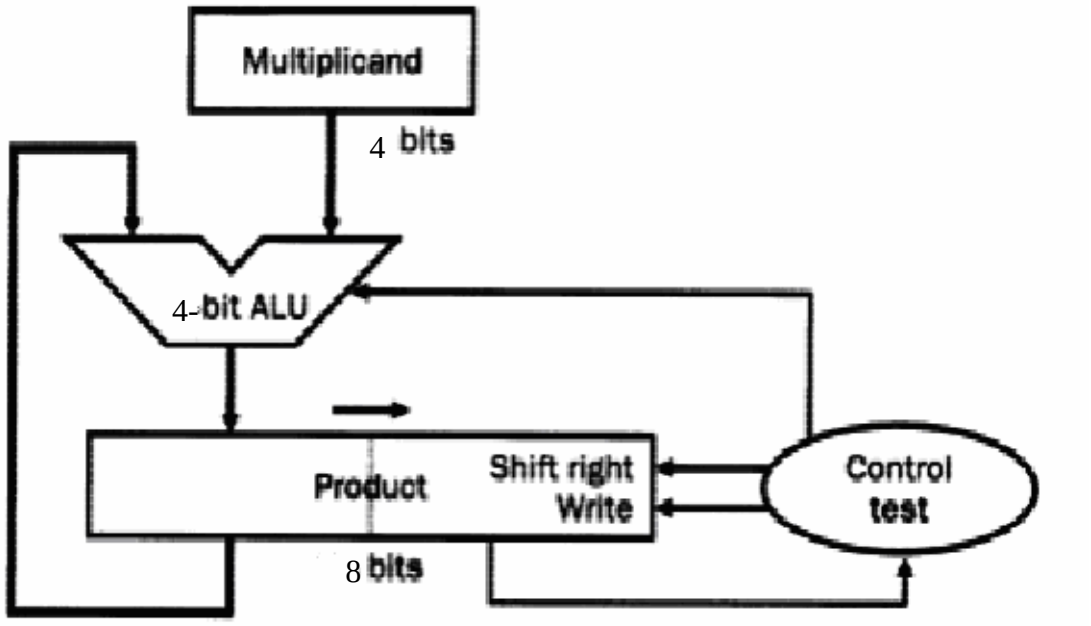
Translate your “C” function from Question 1 into MIPS assembly language, assuming that main() used the following layout to pass arguments on the stack:

```
addi  $sp, $sp, -16
sw    $4, 0($sp)              # multiplicand
sw    $5, 4($sp)              # multiplier
sw    $6, 8($sp)              # pointer to productH
sw    $7, 12($sp)             # pointer to productL
jal   mult32
addi  $sp, $sp, 16
```

Then, translate your "C" main() program into MIPS assembly language to test your binary multiplication function. The SPIM environment includes a number of SYSCALL functions for printing strings and integers.

Test your function with the same arrays of test cases you used for Question 1. Your results should be identical. However, there may be some differences with very large numbers, since SPIM doesn't have a way to print unsigned integers (if SPIM prints a negative number, you can compute the equivalent unsigned positive number by working out the two's complement by hand).

Question 3 (50 points)



The figure shown above shows a datapath for a 4-bit x 4-bit unsigned binary multiplier adapted from the one shown for Question 1. It is easily modified to perform Booth's algorithm by adding a single D flip-flop for bit B-1 (assuming the ALU supports subtraction).

a) Design a 4-bit ALU using the LogicWorks 4-bit full-adder, where F1 and F0 determine the mode of operation as follows. Encapsulate this part using the Device Editor.

F1	F0	Function
0	0	NOP
0	1	Add
1	0	Subtract
1	1	NOP

b) Next design a 4-bit shift register with the following modes of operation. Encapsulate this part using the Device Editor.

M1	M0	Operation
0	0	Hold
0	1	Load 0000 (synchronous clear)
1	0	Shift
1	1	Load

c) Using these parts, design a datapath capable of supporting Booth's algorithm. Make sure to include a signal to clear the D flip-flop at the initialization stage of the algorithm. The complete datapath should include 7 control lines: F1, F0 (ALU), M1high, M0high (Product-high), M1low, M0low (Product-low), and D-init (D-FF).

d) The datapath requires 9 clock pulses to compute the product of two 4-bit signed integers, Determine the complete state transition table corresponding to your controller and draw the corresponding state diagram. Note that the state transition table has 4 state variables (since there are 9 clock pulses), so the following are the headings you'll need.

Q3	Q2	Q1	Q0	Q^3	Q^2	Q^1	Q^0	M1h	M0h	M1l	M0l	D_{Init}

e) Implement your controller using a ROM and register and integrate this with your datapath. To make sense of the output, use two 4-bit registers to copy the output of Product-high (P-high) and Product-low (P-low) every time the state machine returns back to state 0000 (initialization). Use two hex displays to show your output and two hex keypads to enter the multiplier and multiplicand. It is not necessary to implement a timing file here – simply use the LogicWorks clock device and trace P-high and P-low through a complete cycle. To demonstrate that your multiplier works correctly, first work out the contents of P-high, P-low, and B-1 by hand for each of the 9 clock cycles for the case of -7 x 6. Then, enter -7 x 6 into your LogicWorks multiplier, and compare the results to your predictions. They should be the same.

Your submission to this question must be a Microsoft Word document, as well as a folder containing all your circuits. The Microsoft Word document must contain the following:

a) Your 4-bit ALU (copy-and-paste as a graphic from LogicWorks)
b) Your 4-bit Shift Register (copy-and-paste from LogicWorks)
c) Your datapath (copy-and-paste from LogicWorks)
d) Your state transition table and state transition diagram.
e) The contents of P-high, P-low, and B-1 for the case of -7 x 6, worked out by hand, and a screenshot of the LogicWorks timing window, showing the correct computation of -7 x 6.