Name ____

CS/COE 0447 Example Problems for Exam 2
Fall 2010

1. This time, consider a non-leaf function `lisa`. This function has no arugments or return value. The return address is on the stack at offset 12 and the activation record is 12 bytes. Give a sequence of three MIPS instructions that cause a function return.

lw	*$ra,12($sp)*	*# loads the return address*
addi	*$sp,$sp,12*	*# adjust the stack pointer to pop the activation frame*
jr	*$ra*	*# do the return*

2. Suppose the stack pointer (`$sp`) has the value 0xFF0000020 and the activation record has two halfword fields. Give a single instruction that will load the second field in the activation record into register `$t0`.

lh $t0, 2($sp) # current $sp is the AR, 2nd field is in 2nd halfword

For the next questions, consider the program code below (with line numbers):

```
                    # assume $sp = 0xFFFF0020
0                   li    $s0,1
1                   li    $s1,2
2                   jal   _bart
3                   j     quit
4     _bart:        addi     $sp,$sp,-8
5                   sw       $s0,0($sp)
6                   sw       $ra,4($sp)
7                   jal      _homer
8                   lw       $ra,4($sp)
9                   lw       $s0,0($sp)
10                  addi     $sp,$sp,8
11                  jr       $ra
12    _homer:       addi     $sp,$sp,-4
13                  sw       $s1,0($sp)
14                  addi     $s1,$0,10
15                  move     $v0,$s1
16                  lw       $s1,0($sp)
17                  addi     $sp,$sp,4
18                  jr       $ra
19    quit:         ....
```

3. Give the value of `$sp` on each line from the code:

Line 5:	`$sp`'s value is _______*0xFFFF0018*_______________
Line 8:	`$sp`'s value is _______*0xFFFF0018*_______________
Line 13:	`$sp`'s value is _______*0xFFFF0014*_______________
Line 19:	`$sp`'s value is _______*0xFFFF0020*_______________

Name ______________________________

4. Assume memory is all 0s. Fill in the table below to show the memory contents (as words) after the code above executes (i.e., when line 19 is reached):

Address	**Value at this Address**
0xFFFF0028	*0x0*
0xFFFF0024	*0x0*
0xFFFF0020	*0x0 (initial $sp before line 4)*
0xFFFF001C	*address of line 3 (return addr)*
0xFFFF0018	*0x1 ($s0 stored here line 5)*
0xFFFF0014	*0x2 ($s1 stored here line 12)*
0xFFFF0010	*0x0*
0xFFFF000C	*0x0*

note: this solution corrects the error mentioned in class on 11/16/09.

5. Show the steps to multiply the 4-bit numbers 3 and 5 with the "fast shift-add multipler". Use the table below. List the multiplicand (M) and product (P) in binary. In the field "step", write "ADD" when the multiplicand is added. Write "SHIFT" to indicate when the product is shifted. In the iteration "Start" write the initial values for the mutiplicand and product. You may not need all steps (rows) in the table.

Iter.	**Multiplicand (M)**	**Product (P)**	**Step**
Start	*0011*	*0000 0101*	*set product=0s:R*
1	*0011*	*0011 0101* *0001 1010*	*lsb=1 => +M* *shift right 1*
2	*0011*	*0001 1010* *0000 1101*	*lsb=0 => +0* *shift right 1*
3	*0011*	*0011 1101* *0001 1110*	*lsb=1 => +M* *shift right 1*
4	*0011*	*0001 1110* *0000 1111*	*lsb=0 => +0* *shift right 1*
5	*NOT NEEDED*		
6	*NOT NEEDED*		

Iter.	Multiplicand (M)	Product (P)	Step
7	*NOT NEEDED*		

6. Show the steps to multiply an 6-bit number 17 and 3 with Booth's algorithm. Use the table below. List the multiplicand and product in binary. In the field "step", write "ADD", "SUB", or "NO OP" to indicate which operation is done on each iteration.

Iter.	Multiplicand (M)	Product (P)	Step
Start	*010001* *(negation is 101111)*	*000000 00001**1 0***	*set P, with pad bit*
1	*010001*	*101111 000011 0* *110111 10000**1 1***	*lsbs=10: -M* *shift right arithmetic*
2	*010001*	*110111 100001 1* *111011 11000**0 1***	*lsbs=11: +0* *shift right arithmetic*
3	*010001*	*001100 110000 1* *000110 01100**0 0***	*lsbs=01: +M* *shift right arithmetic*
4	*010001*	*000110 011000 0* *000011 00110**0 0***	*lsbs=00: +0* *shift right arithmetic*
5	*010001*	*000011 001100 0* *000001 100110 0*	*lsbs=00: +0* *shift right arithmetic*
6	*010001*	*000001 100110 0* ***000000 110011** 0*	*lsbs=00: +0* *shift right arithmetic*
7	*NOT NEEDED*	*Final answer is:* ***000000 110011***	*N/A*

7. Explain how the "fast shift-add multiply" improves over the original "slow shift-add multiply". Be sure to indicate what hardware changes make the "fast version" faster than the "slow version".

combines registers, reduces size of ALU & does 3-steps in paralle. last two make it fast.

8. Suppose we want to do the computation S = A + B. A and B are positive 2's complement 8-bit binary numbers. Give a boolean expression that indicates whether there was an overflow when these numbers are added. To represent a certain bit *i* in *A*, *B* or *S*, use A_i, B_i or S_i. E.g., bit position 3 in *A* is A_3. Assume the bits are numbered 0 to 7 (right to left).

overflow happens when input values have same sign but output has different one

$Overflow = (A_7 \wedge B_7 \wedge \neg S_7)$ OR $(\neg A_7 \wedge \neg B_7 \wedge S7)$

Name ____

9. Give the negation in one's complement binary representation (5 bit numbers) for the decimal numbers:

5d   Negation (in one's complement binary) ___*11010*____________

10d   Negation (in one's complement binary) ___*10101*____________

-15d   Negation (in one's complement binary) ___*01111*____________

10. Give the negation in two's complement binary representation (5 bits) for the decimal numbers:

11d   Negation (in two's complement binary) ___*10101*____________

15d   Negation (in two's complement binary) ___*10001*____________

-13d   Negation (in two's complement binary) ___*01101*____________

11. Give Booth's encoding for the 8-bit numbers:

-19d   Booth's encoding _______*00-11 0-11-1*_________________

-19 in two's comp: 1110 1101
-19 in two's comp with 0 pad: 1110 1101 0
Booth's encoding: 00-11 0-11-1
check yourself: $-2^5+2^4-2^2+2^1-2^0 = -32+16-4+2-1=-19$

27d   Booth's encoding ________ *0010 -110-1*________________

27 in two's comp: 0001 1011
27 in two's comp with 0 pad: 0001 1011 0
Booth's encoding: 0010 -110-1
check yourself: $2^5-2^3+2^2-2^0 = 32 - 8 + 4 -1 = 27$

62d   Booth's encoding ________*0100 00-10*_________________

62 in two's comp: 0011 1110
62 in two's comp with 0 pad: 0011 1110 0
Booth's encoding: 0100 00-10
check yourself: $2^6-2^1 = 62$

12. Give *two reasons* to use Booth's algorithm (encoding) to improve the multiplication hardware.

1. it can reduce the number of addition operations
2. it handles signed multiplication

13. Floating point numbers represent a "richer" set of values than integer numbers. Nevertheless, processors support integer numbers and programs frequently use them. What primary advantage does integer numbers and operations offer over floating point numbers and operations?

integer operations are significantly faster, programs frequently use discrete values thus, using integer for common operations/values offers a big performance benefit.

14. Using 1-bit adders, draw the circuit for a 4-bit ripple-carry addition unit.

See book / class lecture slides.

15. Using 1-bit adders and 1-bit inverters (i.e., the *not* of a bit), draw a circuit for a 4-bit ripple-carry subtract unit.

See book / class lecture slides (drawn on the board during class).

16. Consider the sum of products boolean equation: A'BC + ABC + A'B'C. Give the truth table representation for this boolean equation. (A' is NOT A and + is OR)

This truth table has eight rows with three input values (A, B, C) and one output. Label the rows by counting in bianry from 0 to 7. Row 011 (i.e., where A=0, B=1, C=1) has a 1 in the output, row 111 has a 1 in the output, and 001 has a 1 in the output. All other output values are 0.

17. Suppose you want to design a hardware circuit that has two inputs A and B, and one output O. The output O has the value 1 when exactly one input (A/B) is a 1. Give the truth table for this circuit design.

INPUTS		***OUTPUT***
A	***B***	***O***
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*0*

(this is exclusive-OR!)

18. For question 15, what is the boolean equation for the truth table?

O = A'B + AB'

Name ____

19. Here is a truth table. What is the boolean equation for O?

INPUTS			**OUTPUT**
A	**B**	**C**	**O**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

O=A'B'C + A'BC' + AB'C'

20. Let's consider a 4-bit adder. If each 1-bit adder takes 2ns to compute an output (1-bit result and carry-out), how long does it take to compute the full 4-bit result?

*it takes 4 * 2ns = 8ns (Note: We'll have more to say about this soon.)*

21. Consider problem 17 again. This time, let's compute the time for subtraction (A-B). Suppose the 1-bit inverter (used to complement the B input of each 1-bit adder) takes 1ns. How long does it take to compute an answer with this subtraction unit? (*Be careful*: Think about whether the invert operations can be done simultaneously.)

it takes 1ns additional for the first invert (9ns total); the rest are done in parallel.

22. Suppose you have a program P. This program executes 1000 regular instructions, 100 floating point multiply instructions, and 5 floating point square root instructions. Assume a regular instruction takes 10ns, a floating point multiply takes 100ns and and floating point square root takes 1000ns. What is the total amount of time (in ns) that the program takes to execute (the "execution time")?

*time = 1000*10ns + 100*100ns + 5*1000ns = 25000ns*

23. Consider the program in question 18. Let's suppose we can improve the floating-point multiply to take 80ns or we can improve the floating-point square root to take 800ns. We cannot do both improvements. Compute the execution time with both improvements. Which improvement would you do?

*timeA = 1000*10ns + 100*80ns + 5*1000ns = 23000ns*
*timeB = 1000*10ns + 100*100ns + 5*800ns = 24000ns*
the first choice is actually faster, so I'd pick this one, assuming "costs" are the same.

24. Now, consider the fastest improvement from 19 and the original situation from problem 18. What is the speedup of the improvement versus the original case?

speedup = slow/fast = 25000ns/23000ns = 1.09 speedup