Name ____

CS/COE 0447 Example Problems for Exam 3
Fall 2009

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 0000**11 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. 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 $(\neg A_7 \wedge \neg B_7 \wedge S7)$

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

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

Name ____

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

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

9. 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*_____________

10. 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: $2^6-2^1 = 62$

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

See book / class lecture slides.

12. 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).

Name ______________________________

13. Consider restoring division with hardware design #3 (the design with a 32-bit divisor and a 64-bit remainder register that holds the remainder and quotient). Assume the quotient and divisor are 5 bit unsigned numbers. Fill in the table below for 17 / 3. For each step, indicate what shift, subtraction, and addition operations are done in the "Step Notes" column.

Iteration	Divisor (D)	Remainder (R)	Step Notes
Init	00011	00000 10001	initial values
1	00011	11101 10001 00001 00010	R = R - D R<0: +D, left shift 0 into lsb
2	00011	11110 00010 00010 00100	R = R - D R<0: +D, left shift 0 into lsb
3	00011	11111 00100 00100 01000	R = R - D R<0: +D, left shift 0 into lsb
4	00011	00001 01000 00010 10001	R = R - D R>0: left shift 1 into lsb
5	00011	11111 10001 00101 00010	R = R - D R<0: +D, left shift 0 into lsb
6	00011	00010 00010 00100 00101	R = R -D R>0: left shift 1 into lsb
Done		00010 00101	right shift the left half of R by 1 result=5, remainder=2

note: the full step 1 is shown for clarity.

14. Now, consider non-restoring division with hardware design #3 (the design with a 32-bit divisor and a 64-bit remainder register that holds the remainder and quotient). Assume the quotient and divisor are 5 bit unsigned numbers. Fill in the table below for 17 / 3. For each step, indicate when shift, addition and/or subtraction operations are done in the "Step Notes" column.

Iteration	Divisor	Remainder	Step Notes
Init	00011	00000 10001	initial values
1	00011	11101 10001 11011 00010	R = R - D R<0: left shift 0 into lsb
2	00011	11110 00010 11100 00100	R = R + D R<0: left shift 0 into lsb
3	00011	11111 00100 11110 01000	R = R + D R<0: left shift 0 into lsb
4	00011	00001 01000 00010 10001	R = R + D R>0: left shift 1 into lsb
5	00011	11111 10001 11111 00010	R = R - D R<0: left shift 0 into lsb
6	00011	00010 00010 00100 00101	R = R + D R>0: left shift 1 into lsb

Name ______________________________________

Done		00010 00101	right shift the left half of R by 1 result = 5, remainder = 2

15. 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.

16. Using IEEE 754 representation for single precision floating point, give the 32-bit binary encoding for the numbers below. Show the sign, exponent, and significand.

Number: -2.40625 Float: ______________________________

answer: sign is -1, so sign bit will be = 1
integral part is 2 = 10b
fractional part:

$0.40625 \times 2 =$	0.8125	1st bit is 0
$0.8125 \times 2 =$	1.625	2nd bit is 1
$0.625 \times 2 =$	1.25	3rd bit is 1
$0.25 \times 2 =$	0.5	4th bit is 0
$0.5 \times 2 =$	1.0	5th bit is 1

thus, the number is 10.01101
we need to normalize the number by shifting the decimal point to the left one position.
in the normalized form, we will shift back to the right, thus the exponent is 2^1
the normalized form is 1.001101×2^1
now, we compute the exponent in biased form: biased exp = 1 + 127 = 128
in binary, the biased exponent = 10000000b
finally, we have the representation:

sign	**exp**	**significand**
1	10000000	0011010...0
1 bit	*8 bits*	*23 bits*

Number: 11.2265625 Float: ______________________________

answer: sign is +, so the sign bit is 0
integral part is 11, so in binary it is 1011b
fractional part is 0.2265625

$0.2265625 \times 2 =$	0.453125
$0.453125 \times 2 =$	0.90625
$0.90625 \times 2 =$	1.8125
$0.8125 \times 2 =$	1.625
$0.625 \times 2 =$	1.25
$0.25 \times 2 =$	0.5
$0.5 \times 2 =$	1.0

thus, we have 1011.0011101

we put into normal form: 1.0110011101×2^3
put the exponent into biased form: biased exp = 3 + 127 = 130
in binary, biased exp = 10000010b
finally, we have the representation:

sign	exp	significand
0	10000010	01100111010...0
1 bit	*8 bits*	*23 bits*

Number: -0.00244140625 Float: ____

answer: sign is -, so the sign bit is 1
integral part is 0, so in binary it is 0b
fractional part is 0.00244140625

$0.00244140625 \times 2 =$	0.0048828125
$0.0048828125 \times 2 =$	0.009765625
$0.009765625 \times 2 =$	0.01953125
$0.01953125 \times 2 =$	0.0390625
$0.0390625 \times 2 =$	0.078125
$0.078125 \times 2 =$	0.15625
$0.15625 \times 2 =$	0.3125
$0.3125 \times 2 =$	0.625
$0.625 \times 2 =$	1.25
$0.25 \times 2 =$	0.5
$0.5 \times 2 =$	1.0

thus, we have 0.00000000101
we put into normal form: 1.01×2^{-9}
put the exponent into biased form: biased exp = -9 + 127 = 118
in binary, biased exp = 01110110b
finally, we have the representation:

sign	exp	significand
1	01110110	010...0
1 bit	*8 bits*	*23 bits*

17. Suppose we have the following numbers encoded as IEEE 754 single precision floats. is the decimal value (i.e., base 10) for each number:

sign	exp	significand
0	10000001	011010...0
1 bit	*8 bits*	*23 bits*

answer: sign bit is 0, so it's positive
biased exp = 10000001b = 129, so actual exp = 129 - 127 = 2
thus, we have 1.01101×2^2
moving the decimal point to the right 2 places, we get: 101.101b
integral part is 101b = 5d
fractional part is 101b = $2^{-1} + 2^{-3} = 0.625$

so, the answer is 5.625d

sign	exp	significand
1	01111011	10...0
1 bit	*8 bits*	*23 bits*

answer: sign bit is 1, so it's negative
biased exp = 01111011b = 123, so actual exp = 123 - 127 = -4
thus, we have 1.1×2^{-4}
moving the decimal poit to the left 4 places, we get 0.00011
integral part is 0
fractional part is $2^{-4} + 2^{-5} = 0.09375$
so, the answer is -0.09375

18. When the bias is 127, give the binary encoding for the number 39d.

biased number = 39 + 127 = 166d = 10100110b

19. When the bias is 1023, give the binary encoding for the number -39d.

biased number = -39 + 1023 = 984d = 1111011000b

20. Biased representation for the exponent offers a significant advantage over other representations, like two's complement. What is the advantage?

they allow the use of integer (faster) operation to do simple sorting based on the exponent

21. Consider the sum of products boolean equation: A'BC + ABC + A'B'C. Give the truth table representation for this boolean equation.

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.

22. For the boolean equation and truth table from question 21, give its Karnaugh map.

23. Based on the Karnaugh map from question 22, can the boolean equation be minimized? If so, give its minimized form. Otherwise, simply list the original equation to answer this question.

24. For your answer from question 23, draw the circuit that implements the boolean equation.

Name ______________________________

25. Give the minimized boolean equation for the Karnaugh map below.

		CD			
		00	**01**	**11**	**10**
AB	**00**	0	0	0	0
	01	0	1	1	0
	11	0	1	1	0
	10	0	0	0	0

In AB (01 and 11), A is both 0 and 1, so it can be eliminated. In CD (01 and 11), C is both 0 and 1, so it can be eliminated. Thus, we have just BD for this equation.

26. Give the minimized boolean equation for the Karnaugh map below.

		CD			
		00	**01**	**11**	**10**
AB	**00**	0	1	0	1
	01	0	1	0	1
	11	0	1	0	0
	10	0	1	0	0

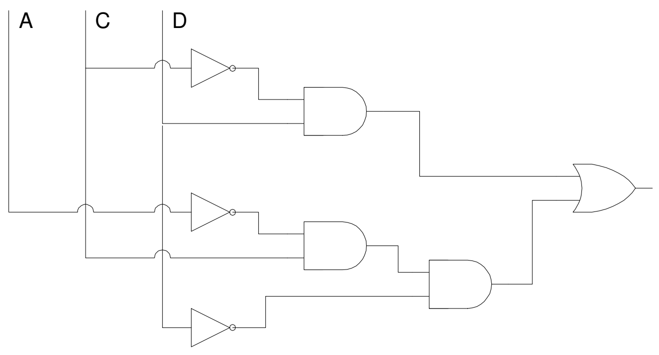
Look at the column for C′D (01): A and B have both 0 and 1, so they can be eliminiated, giving only C′D. Now, consider the column for CD′ (10): B is both 0 and 1, thus it can be eliminated. This minterm is A′CD′. The minimized sum of products is: C′D + A′CD′.

27. Suppose we want to construct a 4:1 multiplexor. This multiplexor selects as an output one of its four inputs. How many rows are in the full truth table for this operation?

A 4:1 mux has six input signals. There are four input "data signals" (to select among) and two "control signals" that select one of the four input data signals to steer to the output. Thus, the full truth table has 2^6 rows.

28. Using AND, OR, and NOT gates, draw the circuit for the boolean equation from question 26.

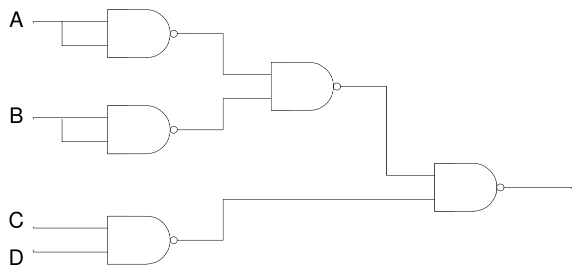
Circuit for C′D + A′CD′



Name ____

29. Using only NAND gates, draw a minimal circuit for the boolean equation A'B' + CD. Hint: Use straightforward substitutions of ANDs, ORs and NOTs with the NAND equivalents (see lecture slides). Observe whether any of the operations after substitution can be eliminated.

Simplified circuit for A'B' + CD (using NAND gates)



30. Prove that your circuit from question 29 is equivalent to A'B' + CD. Hint: Write a boolean equation for the circuit and transform it, step-by-step, into A'B' + CD with Boolean algebra.

The equation for the circuit is ¬(¬(¬A ^ ¬B) ^ ¬(C ^ D)). We can transform this as:

¬(¬(¬A ^ ¬B) ^ ¬(C ^ D))	*Initial equation*
= ¬((¬¬A v ¬¬B) ^ (¬C v ¬D))	*Distribute inner nots with DeMorgan's Law*
= ¬((A v B) ^ (¬C v ¬D))	*Simplify the "not-not"*
= ¬(A v B) v ¬(¬C v ¬D)	*Distribute outer not with DeMorgan's Law*
= (¬A ^ ¬B) v (¬¬C ^ ¬¬D)	*Distribute not with DeMorgan's law*
= (¬A ^ ¬B) v (C ^ D)	*Simplify the "not-not" and we're done.*

thus, we have arrived at A'B' + CD (using different notation).

31. Using only NOR gates, draw a minimal circuit for the boolean equation A'B + CD. (Can you prove this circuit is equivalent to A'B' + CD?)

This exercise is left to the reader to solve. (Do the substitution as in problem 17 and then simplify the circuit with Boolean algebra, if possible.)

32. Suppose we want to implement a 1-bit ALU that can perform the logical operations OR, AND, XOR and NOT. This 1-bit ALU takes as input two 1-bit numbers: A and B. It produces as an output a single 1-bit number: C. If the ALU does a unary NOT operation, it ignores B. To build this ALU, you can use OR, AND, XOR and NOT gates. You also need a multiplexor to select among the four operations. Answer the following questions:

a) How many control signals are needed to select the operation to do with the ALU?

2 bits (signals) to select one of the four operations

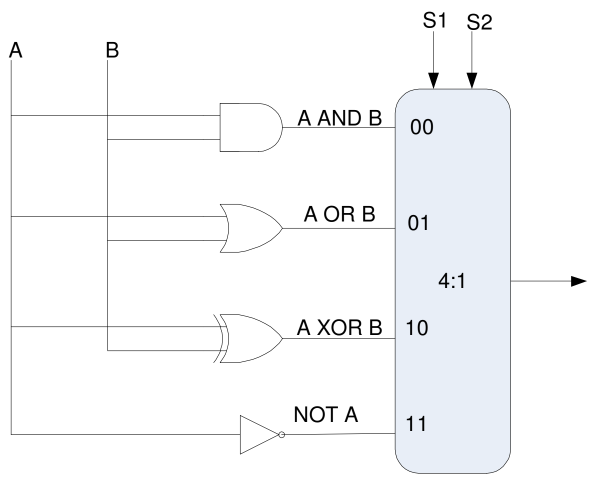
b) How many inputs does the ALU's multiplexor have?

6 inputs: 4 values from OR, AND, XOR, and NOT gates, plus 2 control signals

Name ____________________

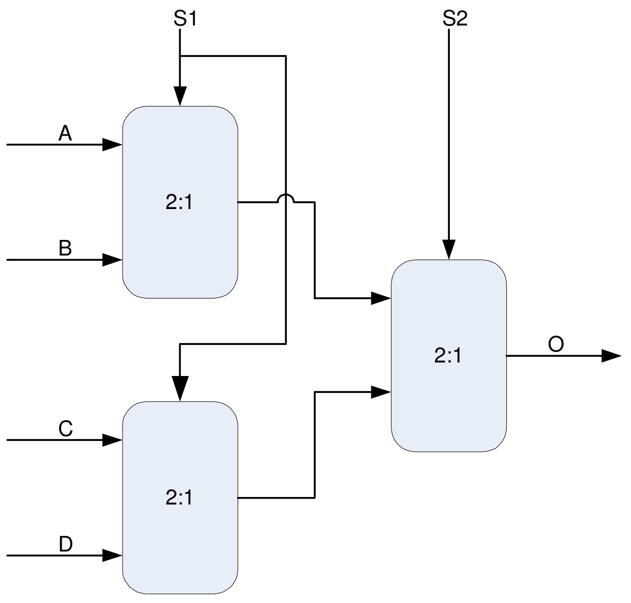
c) Draw the ALU circuit (using symbols for a mux, AND, OR, NOT and XOR).

Four Function 1-bit ALU



33. Larger muxes can be constructed from smaller ones. For example, a 4:1 mux can be constructed with 2:1 muxes. A 4:1 mux selects among four inputs for its one output. Draw a circuit diagram for a 4:1 mux, built with 2:1 muxes. (Hint: You need three 2:1 muxes.)

4:1 mux built with 2:1 muxes (S1 and S2 are select signals)



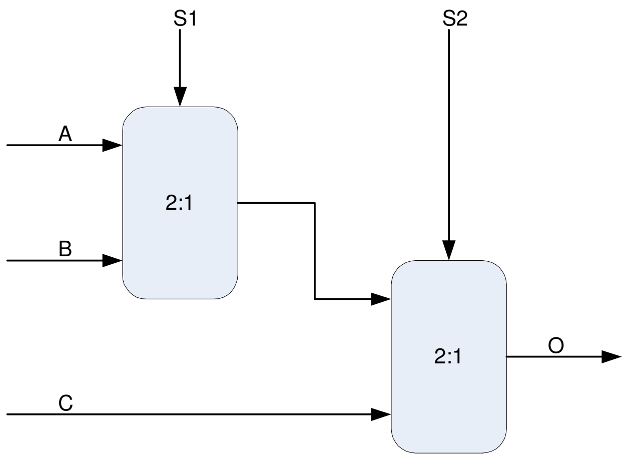
S1 and S2 select one of the four inputs A, B, C, or D to steer to the output O. S1 and S2 is essentially a 2-bit selector: value 00 (S2=0, S1=0) selects A, 01 (S2=0, S1=1) selects B, 10 (S2=1, S1=0) selects C and 11 (S2=1, S1=1) selects D.

Note: There are other ways to build muxes, but this is the most straightforward.

Name ___________________________________

34. Now, let’s consider a 3:1 mux built with 2:1 muxes. Draw a diagram that shows the circuit. (Hint: You need two 2:1 muxes.)

3:1 mux built with 2:1 muxes (S1 and S2 are select signals)



Note: The selection values (i.e., signal values for S1 and S2) are not all used. In particular, the values used are 00 (select A), 01 (select B), and 1X (select C, where X is “don’t care” for S1).

35. Suppose a logic gate (OR, AND, NOT) takes 1 ns to compute a value. The time it takes a gate to compute a value is a “delay”. If multiple gates are put together in succession, the total time for the combinational circuit will be the sum of the gate delays. For example, a circuit that has an AND gate followed by an OR gate has a delay of 2ns (1ns for each gate). As another example, a circuit that has an AND gate that operates in *parallel* with an OR gate has a delay of 1ns (since the OR and AND are done in parallel). If each gate takes 1ns, compute the total delay for a 16:1 mux built with 2:1 muxes. Hint: Compute the time for a single 2:1 mux. Next, determine how many muxes are connected in succession. The “longest path” through this circuit gives the total delay.

*First, note that the longest path in a 2:1 mux has three gates: NOT -> AND ->OR (see diagram in logic1.pdf, slide #26 on page 13). Thus, a 2:1 mux has a 3 * 1ns = 3ns delay.*

Second, let’s compute the number of 2:1 muxes that have to be traversed in a 16:1 mux. This structure is similar to the one from problem 21, except it has more “levels”. There are four levels: level 1 has 8 muxes, level 2 has 4 muxes, level 3 has 2 muxes and level 4 has 1 mux. To traverse the full 16:1 mux, one mux in each level has to be traversed.

*Finally, with four levels and 3ns per mux (level), the total delay is 4 * 3ns = 12ns.*

Note the relationship to binary trees!