

A7: Digital Logic

“The Pokemon of Logic”

Due: Wed May 27, 2020 @11:59PM

Answer each problem in the boxes provided. Any writing outside of the boxes will NOT be graded. Do not turn in responses recorded on separate sheets.

- Handwritten or typed responses are accepted. In either case, make sure all answers are in the appropriate boxes and should be filled out in this document whether this is done by printing or editing in a PDF editor. If you are unable to scan or print, then it is alright to write your answer on a separate sheet of paper, but please make the questions obvious. Here is a link for scanning:
http://gradescope-static-assets.s3-us-west-2.amazonaws.com/help/submitting_hw_guide.pdf
- All responses must be neat and legible. Illegible answers will result in zero points.
- **DO NOT** take a picture of your computer screen
- **DO** make sure to assign the questions to the correct gradescope questions or there will be a point deduction
- Make sure your responses are visible on Gradescope! If they aren't they will NOT be graded!

Learning Goals

- ARM Assembly sp and fp
- De Morgan's Law
- Logic gates
- Truth Tables
- Circuits
- Ripple carry adder

Question 1 (13 points)

A. Given the following scenario where you initially called the function `func`, what is the value of the `fp` at label `two` and the value of `sp` at the label `two`? Assume the value of the `sp` *after* the push in `func` is `0xfffff1400` (i.e. when you are about to branch to `one`, `sp` is `0xfffff1400`). Express your answer in hexadecimal. **Hint:** It helps to draw the stack frame

```
one:    push    {r4-r10, fp}
        add     fp, sp, #28

two:    nop                                // nop means no operation
        add     sp, sp, #-4               // allocate local variable space
        add     r4, r0, r1               // r4 = r0 + r1
        str     r4, [fp, #-32]           // varA = r4
        add     r5, r2, r3               // r5 = r2 + r3
        ldr     r1, [fp, #-32]           // r1 = varA
        sub     r0, r5, r1               // result = r5 - varA
        sub     sp, fp, #28
        pop     {r4-r10, fp}
        bx      lr

func:   push    {lr}
        bl      one                     // current sp is 0xfffff1400 at this line
        pop     {lr}
        bx      r14
```

Answer	
sp =	0x
fp =	0x

B. Given the code in the first part, what is the *minimal* set of registers that must be saved by the routine `one` if one ignores the need for 8 byte stack alignment?

Answer

C. Given the code in the first part, and the “minimal” set of registers that must be saved if one ignores the need for 8 byte stack alignment, what should the instruction `add fp, sp, #28` be changed to?

Answer

D. Why must `lr` be pushed and popped under the label `func`? (consider what would happen if `lr` were not pushed).

Answer

E. Why isn't `lr` pushed and popped under the label `one`?

Answer

F. Why aren't we pushing `r11` for the label `one` (or are we...) ?

Answer

Question 2 (4 points; 0.5 points per cell in the last column)

The pokemon Missingno is capable of learning a new pokemon move to solve complex boolean logic. However, when they are first starting out they are prone to errors and need their pokemon trainers to help correct them. Help train Missingno by filling out the truth table below for the following expression:

$$f(x, y, z) = ((x' \text{ xor } y) (x' z)' + y)'$$

Note that the following are equivalent syntax: x' , $\sim x$, $\text{not}(x)$. You don't need to fill out the columns in between, we only need the end truth (f is sufficient)! However, filling them out may be helpful for figuring out column f (the truth).

x	y	z	$(x' \text{ xor } y)$	$(x' z)'$	$(x' \text{ xor } y) (x' z)'$	f
1	1	1				
1	1	0				
1	0	1				
1	0	0				
0	1	1				
0	1	0				
0	0	1				
0	0	0				

Question 3 (5 points)

Use DeMorgan's Law to show that:

$$z = ((a' + b') \cdot c' \cdot d')' \text{ is the same as } z = ab + c + d$$

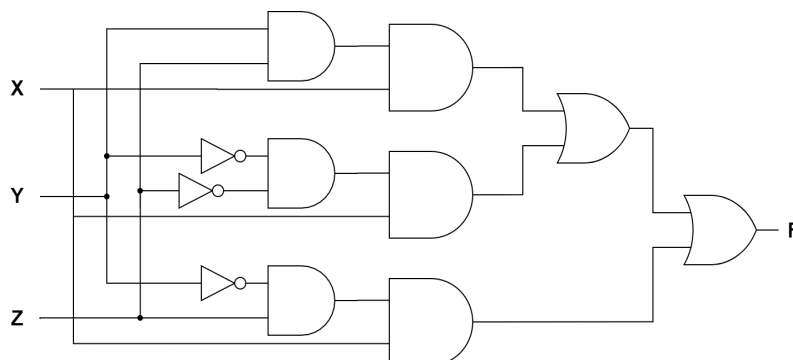
Show work

Question 4 (8 points)

A) Your team at Poketch Inc. is ever-so-grateful to you for fixing up their backend in Memwrap, and have promoted you to Junior Poketch Manager. However, now you realise that the watches are incredibly bulky and need to be charged frequently because it consumes a lot of power to operate leading to frustrated Pokemon Trainers waiting in long lines at their nearest Pokemon Center to charge their watches. Your team turns to you again to fix this problem so that young Pokemon Trainers all over wouldn't have to make frequent stops just to charge their Poketches.

You look at the schematics of their backend hardware and, to your horror, find that there are plenty of redundant gates and wires and convoluted logic all over the place. If you can make their circuits more efficient, the Poketchs will become smaller and require lesser power. Soon enough you'll be on your path for another promotion!

One such component implements the boolean expression: $(xy)z + x(y'z') + (xy')z$, and its circuit is shown below. **Simplify this circuit down to THREE gates and report the simplified expression.**



Hints:

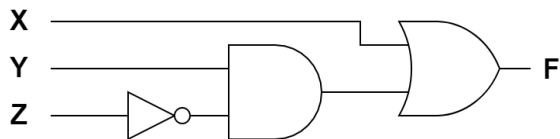
- Factor out a common term.
- Realize that $x + x' = 1$ and
- hence $a + a'b = a + b$
- One of the 3 gates is an inverter

Show work here for partial credit

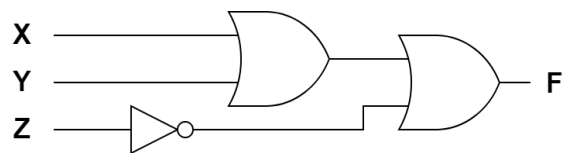
Answer

B) Which of the following circuit diagrams represents the simplified expression obtained in Part A? Hint: make a truth table for each to confirm that they are equivalent.

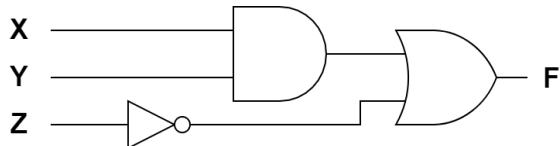
A.



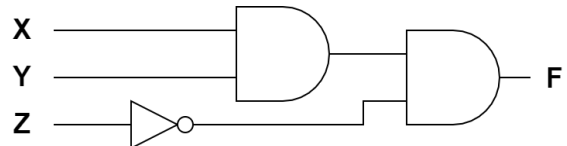
D.



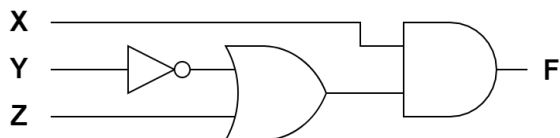
B.



E.



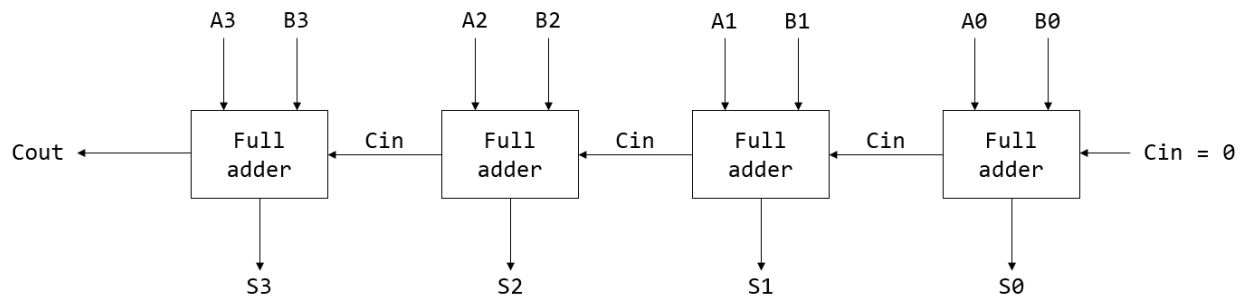
C.



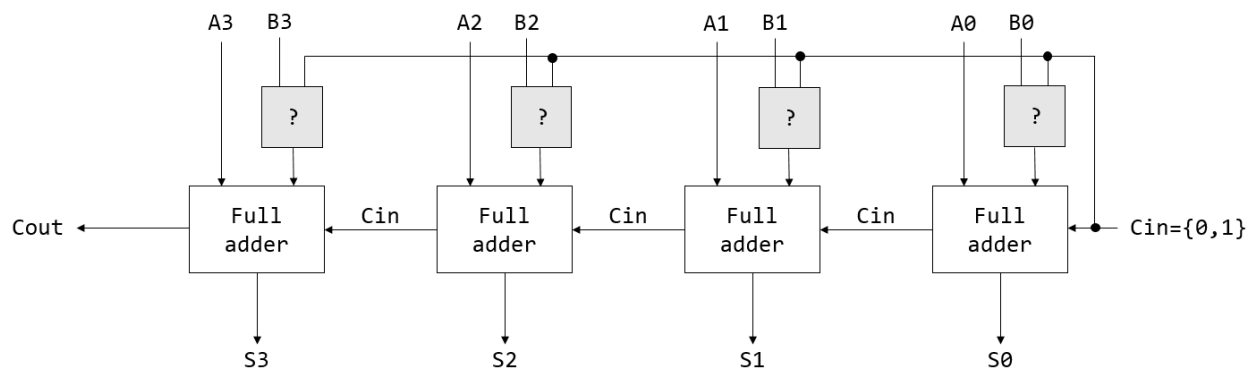
Answer

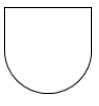
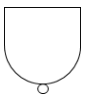
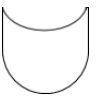
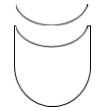
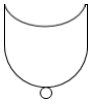
Question 5 (4 points)

Below we have a diagram of a 4-bit ripple carry adder.



What if we wanted to modify this to also handle subtraction? Instead of the initial carry in (C_{in}) value for the first bit adder being 0, we can imagine it being either a 1 or a 0. If $C_{in} = 0$, then our adder will perform addition, but if $C_{in} = 1$, then our adder will perform subtraction. What is the appropriate gate that belongs in the “?” boxes below to achieve this functionality?



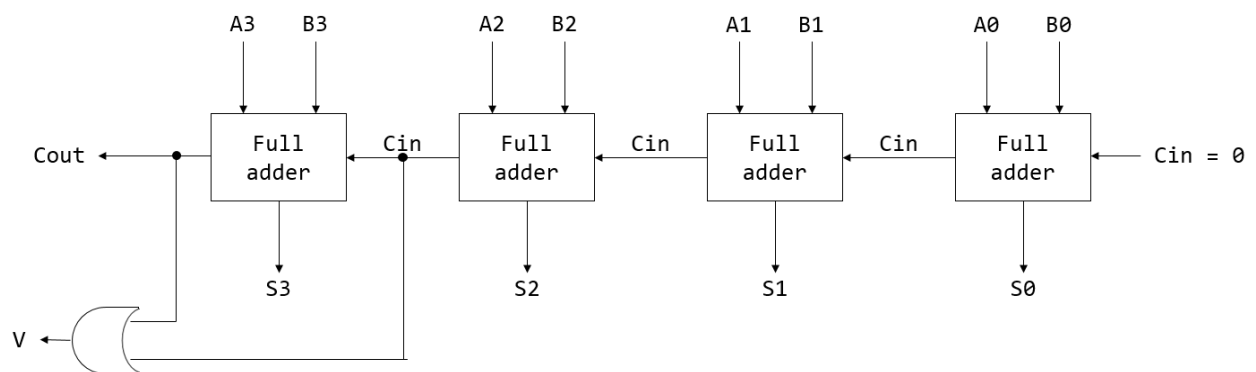
A. 	B. 	C. 
D. 	E. 	

Answer

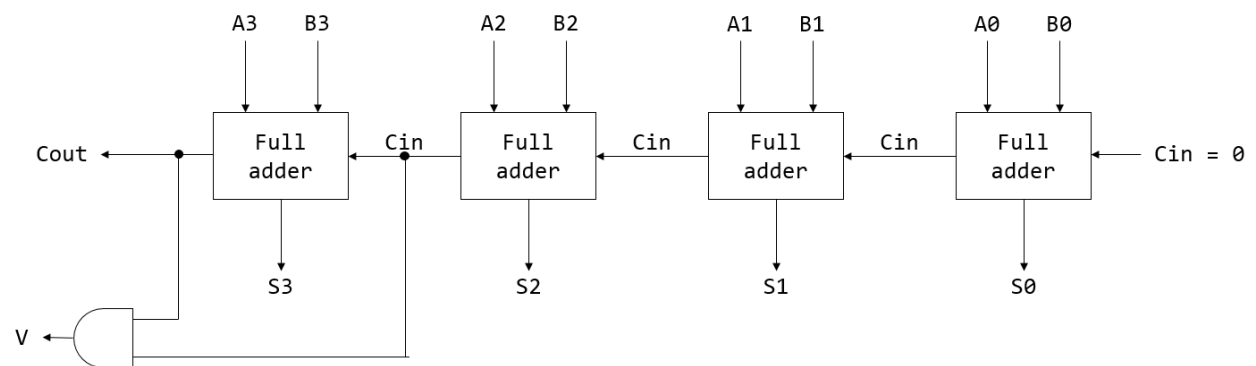
Question 6 (8 points)

We want to detect whether or not overflow occurred as a result of adding two positive numbers? We can modify our simple 4 bit adder from before to capture overflow output. Which of the following diagrams would capture overflow from *signed* addition of two positive numbers and which would capture overflow from *unsigned* addition for two positive numbers? (See the 4 diagrams below)

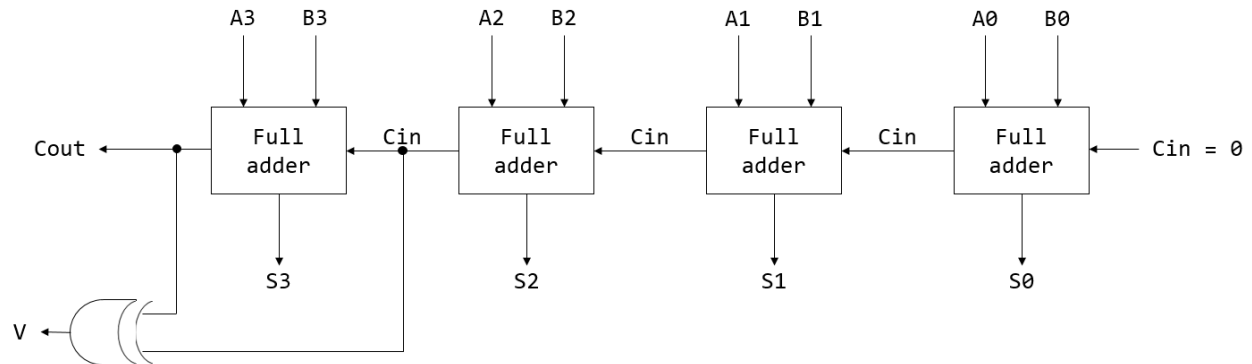
Circuit 1:



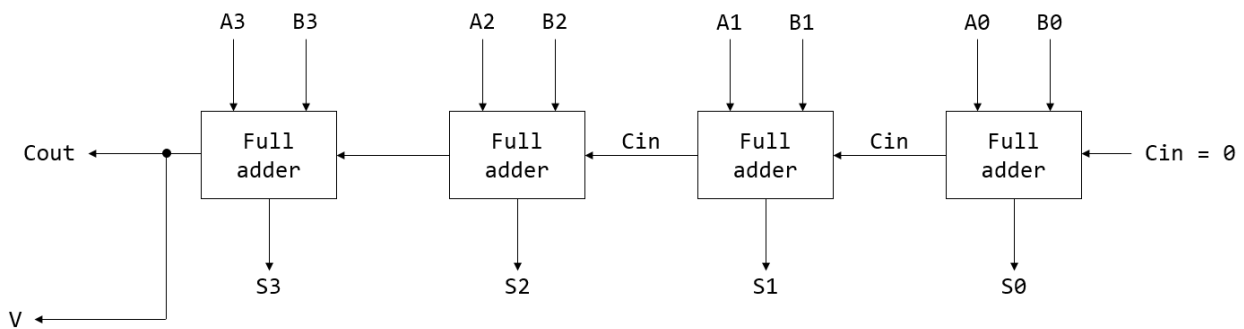
Circuit 2:



Circuit 3:



Circuit 4:



- A. **Circuit 2** handles overflow for signed addition, and **Circuit 1** - for unsigned
- B. **Circuit 2** handles overflow for signed addition, and **Circuit 4** - for unsigned
- C. **Circuit 3** handles overflow for signed addition, and **Circuit 4** - for unsigned
- D. **Circuit 3** handles overflow for signed addition, and **Circuit 3** - for unsigned
- E. **Circuit 3** handles overflow for signed addition, and **Circuit 2** - for unsigned

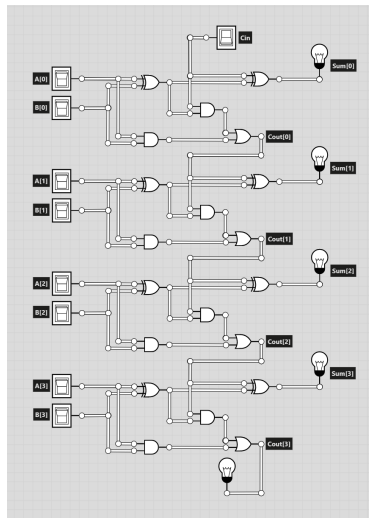
Answer	
---------------	--

Question 7 (8 points)

We will now examine the parallel nature of hardware through a simple 4-bit adder, using the interactive circuit simulation website **logic.ly**!

1) Open www.logic.ly/demo. Close the popup box if it appears.

2) Open the **pastebin** link <https://pastebin.com/2HCSPk3J>, copy the full **RAW Paste Data** string contained inside, and press Ctrl-V (i.e. paste it) inside logic.ly. The logic.ly website encodes its circuits in raw string form, so you can simply copy-paste them as strings! You should see the following circuit appear:



3) Now before trying each of the input combinations, **you must first get the circuit into a state where every wire has the logical value of 0 on it.** (Note: grey colored wires have undefined values. White is 0. Blue is 1.) To do this:

- i) Press *Simulate* → *Reset Simulation* (sets switches to 0 ie white wire is 0)
- ii) Press *Simulate* → *Resume Simulation* (lets the 0s propagate through wires)
- iii) Press *Simulate* → *Pause Simulation* (allows you to simulate step-by-step)

4) Toggle the switches to set the A and B inputs bit-by-bit (down is **logic 1**, up is **logic 0**)

5) Iteratively click *Simulate* → *Advance Simulation One Step*, and you will now see the logic propagate through the circuit **in parallel**

For each the following input combinations (in 2's complement), reset the circuit (step 3), set the A and B inputs (step 4), iterate through the logic (step 5), and **report the number of times you had to iterate until the output reached a stable state that no longer changed.**

For example if you do $3+4$. Then the number of steps it takes to reach a stable state (ie. a state where things no longer change) is 3.

a) How many simulate steps does it take to add $5 + (-6)$?

Answer	
---------------	--

b) How many simulate steps does it take to add $4 + 1$?

Answer	
---------------	--

c) How many simulate steps does it take to add $5 + 7$?

Answer	
---------------	--

d) How many simulate steps does it take to add $(-3) + 4$?

Answer	
---------------	--

Extra: For fun (and further understanding) what does the Cin input do? Add $3 + 4$ and set Cin to 1 and simulate. What is the result?