CPSC 121 2016W2 Final Exam Sample Solution

May 10, 2017

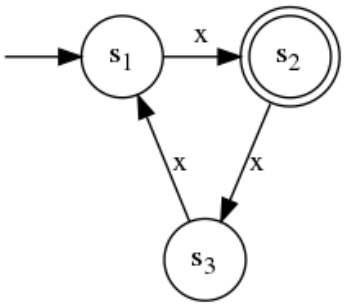
NOTE: There are many correct solutions to some of these problems. These sample solutions are provided to help you understand sample correct answers. **PLEASE WORK THE PROBLEMS CAREFULLY YOURSELF** before reviewing the sample solutions, and critically compare your answers to the solutions with help and feedback from others.

1 Short ∧ (Maybe) Quick [27 marks]

1. **Draw a DFA** that meets the requirements below. **[3 marks]**

   - exactly three states: s_1, s_2, and s_3,
   - exactly one letter in its input alphabet: x,
   - the start state s_1,
   - a single accepting state s_2, and
   - **a transition function that is injective** (also known as "one-to-one")

   SOLUTION: There are several possible, but they must look roughly like this:



   Other than meaningless variations (like moving things around or scaling them up or down), the only real differences can be in where the arrows go. Critically, each of the three arrows must end on a different state. (So, if there is any self-loop—which is OK but strange and unlikely to be seen in student responses—it means that at least one state in the DFA must be unreachable.)

2. Consider the following (non-contradictory) premises:

   - $b \oplus q$
   - $\sim r \rightarrow (p \wedge \sim q)$
   - $r \wedge s$
   - $(a \vee q) \rightarrow \sim s$

Circle each of the following expressions that must also be true, given that these premises are true: **[3 marks]**

SOLUTION: The exactly three correct answers are:

- b (because s means the negation of $a \vee q$ is true, which gives us $\sim q$, which gives us b)
- $\sim a \vee \sim r$ (because we have $\sim a$ by some similar steps to the above and can "or" on anything we like to it)
- $\sim p \vee (p \wedge \sim q)$ (because we have $\sim q$ and the rest of this statement is just playing about with logical equivalences... try applying distribution to this expression to see why)

As for the incorrect answers:

- Since the premises are non-contradictory and b follows, we know that $\sim b$ does not follow. (Furthermore, the only way we could establish $\sim b$ is by establishing q, and we have no statement we could use to establish q. At best, the conditionals can only establish $\sim q$, the first if we can prove its left side and the second if we can disprove its right side.)
- $\sim (p \wedge \sim q)$ is logically equivalent to $\sim p \vee q$. As discussed above, we **can** establish $\sim q$, which means our only hope of establishing this is to establish $\sim p$. No statement we have gives us any hope of establishing that. (Only the second statement mentions p, and it could be used only to establish p—if we could establish its left-hand side—but not $\sim p$.)
- The easiest statement to rule out is $u \rightarrow w$ because neither u nor w is mentioned by any of the premises. Thus, we know nothing about their truth values. Since $u \rightarrow w$ is clearly not a contradiction or a tautology—it can be true, e.g., if u is false and false if u is true and w is false—then neither this statement nor its negation follows from the premises. (Note: we're relying on the fact that the premises are non-contradictory. **EVERY** statement follows from contradictory premises!)

3. **Directly translate the following into propositional logic**: "The result is correct exactly when the first item had the best score but the result does **not** build on the first item's result."

   Use the definitions: $r \equiv$ the result is correct, $s \equiv$ the first item had the best score, and $b \equiv$ the result builds on the first item's result. **[2 marks]**

   SOLUTION: $r \leftrightarrow (s \wedge \sim b)$. (Dropping the parentheses is OK.) We also accepted $(r \leftrightarrow s) \wedge \sim b$ which is a plausible reading of the English statement (i.e., if you imagine a pause after the words "had the best score").

4. Assume that E is the set of all "items" and that we use the function $s : E \rightarrow \mathbb{N}$ to compute scores. That is, s maps each item to its score, a natural number (0, 1, 2, 3, ...). **Define a new predicate** $B(e)$ that means that item e has the **unique** largest score, in the sense that nothing else is tied for largest with e. **[3 marks]**

   SOLUTION: There are definitely some variations possible, but let's start with a good answer:

   $$B(e) \equiv \forall e_2 \in E, e \neq e_2 \rightarrow s(e_2) < s(e)$$

   The natural variation is to use a negated existential instead, in which case the body of the quantifier would also be negated. It's also fine to use a $\leftrightarrow$ rather than a $\rightarrow$ in this case (because we do know that if $s(e_2) < s(e)$, then e_2 must not be e because e can have only one score!). (It's **not** OK to use a $\leftarrow$ where we have a $\rightarrow$, however.)

   Note that as always, it is **NOT** OK to quantify a parameter to the predicate, in this case e. (This is similar to writing a function in a programming language in which you declare a local variable with the

same name as a parameter. That's legal in most programming languages but often quite problematic! It "shadows" or hides the parameter so that you can no longer access it. The semantics in logic are the same: you no longer have access to the parameter.)

5. Using the same definitions, **translate the following to clear, concise English**: **[2 marks]**

   $\forall e_1 \in E, \forall e_2 \in E, |s(e_1) - s(e_2)| < 1$

   SOLUTION: This is surprisingly intricate, and we gave substantial partial credit for on-track answers.

   A literal translation runs something like "for every item and second item (which may be the same as the first), the absolute difference between the score of the first and the score of the second item must be less than 1".

   We can do much better with "the difference between the scores of any two items is less than 1". (We needn't muck with the "which may be the same" part because we **know** if they're the same their scores' difference is obviously less than 1; it's 0 because one item always has its own same score, no matter how many times you call s.) If this statement seems a bit ambiguous in that it perhaps suggests that **all** scores fall in a total range of 1 (e.g., perhaps they range from 3.5 up to 4.5), that's because it **does** imply that. Since the scores all fall on a number line, it must be the case that all the scores are in a single "1-point" range on that line or the statement would be violated by the two items that define the extreme points.

   In fact, however, because scores are natural numbers (see the co-domain of the function s above), this actually says "all items have the same score", which is certainly the clearest and most concise statement.

6. Still using the same definitions and assuming that a and b are two different elements of E, consider the following expression. (Note the lack of absolute value bars.)

   $(\sim\exists e_1 \in E, \exists e_2 \in E, e_1 \neq e_2 \wedge s(e_1) - s(e_2) < 2) \wedge s(a) - s(b) = 4$

   (a) **Is the expression a tautology, contradiction, or contingency**? (Note: we're asking about this expression **alone**, not assuming any expressions from previous questions.) **[1 mark]**

   SOLUTION: CONTRADICTION

   That's because if $s(a) - s(b) = 4$, then $s(b) - s(a) = -4$, but $-4 < 2$. So, $e_1 = b$ and $e_2 = a$ provides a witness to the truth of $e_1 \neq e_2 \wedge s(e_1) - s(e_2) < 2$, which falsifies the LHS of the statement. That means that if the RHS of the statement is true, the LHS is false, which makes the whole statement false. And, if the RHS is false, then the whole statement is **also** false.

   In other words, the absolute value bars are critical!

   (b) Now, **write a logically equivalent expression to the expression but with its negation moved as far inward as it can go**. **[2 marks]**

   SOLUTION: $(\forall e_1 \in E, \forall e_2 \in E, e_1 = e_2 \vee s(e_1) - s(e_2) \geq 2) \wedge s(a) - s(b) = 4$

   Note that the $s(a) - s(b) = 4$ part is unambiguously outside of the negation. So, we flip the quantifiers, change $\wedge$ to $\vee$, and change $\neq$ to $=$ and $<$ to $\geq$. ($x > y$ is **not** the negation of $x < y$.)

7. Knowing only that $a \in E$, $b \in E$, and $a \neq b$ and that A contains the 26 lower-case English letters ($\{a, b, c, d, \ldots, x, y, z\}$), **state clearly what we know about the cardinality (size) of each of the following**. The first one is given as an example. **Leave your answers as clear mathematical expressions** since some would be hard to calculate by hand! **[5 marks]**

   SOLUTIONS:

(a) **Solved example:** E. *Solution:* $|E| \geq 2$. (That is, the cardinality of E is at least two.)
We intended this solved example to guide people in format and reasoning for the rest of the problem. Many students nonetheless struggled with that (e.g., not discussing cardinality or believing that they could give a particular number as the cardinality of each answer). Hopefully, the lesson is to read the question carefully.

(b) $|\mathcal{P}(E)| \geq 2^2$ (and $2^2 = 4$)

(c) $|\mathcal{P}(A)| = 2^{26}$

(d) $2 \leq |E \cap A| \leq 26$. This was the trickiest question. You know that E and A share at least two items in common (a and b). Most people saw that. You **also** know that E and A share at most all the elements of A in common, no matter how big E is. (We missed that part until we were polishing the marking scheme by reviewing student answers!)

(e) $|A \times A| = 26^2$

(f) $|A \cup A| = 26$. (And indeed, $A \cup A = A$.)

8. A string "hash function" maps strings to natural numbers (integers greater than or equal to zero). Imagine that you know for the hash function h that $h(\text{"hello"}) = 7$ and $h(\text{"goodbye"}) = 7$, although h is also defined for all other possible strings.

(a) What is the domain of h? **[1 mark]**
SOLUTION: the set of strings

(b) Is h **injective** (also known as "one-to-one")? **[1 mark]**
SOLUTION: NO because two pre-images map to the same image. (We can see this from the only two mappings given in h.)

(c) Is h **surjective** (also known as "onto")? **[1 mark]**
SOLUTION: NOT ENOUGH INFO because we don't know if every natural number is mapped. For example, if $h(\text{"}n\text{"}) = n$ for all string representations of natural numbers n, then regardless of what else happens, the function **is** surjective. On the other hand, if nothing maps to 0, the function isn't surjective.

(d) Is h **bijective** (also known as a "one-to-one correspondence")? **[1 mark]**
SOLUTION: NO. More specifically, the answer is **NO** if either (or both) of the previous two answers is **NO** (which is the correct answer). It's **YES** if both of the previous two answers is **YES** (which is correct **given** incorrect answers above). The answer **cannot** otherwise be **YES** (because a **YES** answer means we know the answers to the previous two problems). The answer **cannot** otherwise be **NO** unless both answers above are **NOT ENOUGH INFO**. However, if both answers above are **NOT ENOUGH INFO**, one could theoretically imagine being able to tell the function is not bijective without knowing why it's not bijective (i.e., because it's not surjective or because it's not injective). We didn't allow that rather strange response (which would have to be based on already-incorrect work above) in our marking scheme, however.

9. We are proving by induction that Ryan can buy any integral number of hot dogs greater than or equal to 7 by buying a collection of 4-, 5-, 7-, and 9-hot dog packages.

(a) Imagine that the inductive step of our proof shows that if we can buy n hot dogs with these packages, then we can buy $n+7$ hot dogs with these packages. **Complete the portion of the inductive step of the proof after the following sentence. [1 mark]**
"By the induction hypothesis, we can buy packages adding up to n hot dogs. So, ..."
SOLUTION: "we can add one 7-hot dog package to the collection and buy $n+7$ hot dogs" (or the like).

Note that the problem statement clearly establishes that we're "building from" n and "building to" $n+7$. That's not a typical induction argument, but it's unambiguously where this one is going. If you don't understand that this is the answer or if you do understand that this is the answer but don't understand that such an argument could be legitimate, be sure to work it through with someone! Not understanding that suggests your knowledge of how and why induction works is "fragile".

(b) If we were to **change** the inductive step to show instead that if we can buy n hot dogs with these packages, then we can buy $n+5$ hot dogs with these packages, **would the number of base cases we need increase, decrease, or stay the same**? **[1 mark]**
SOLUTION: DECREASE. We needed 7 base cases before to avoid "reaching past" the base cases. Now, we need only 5.

2 Fill-in-the-Blank Induction [7 marks]

Consider the theorem that for integers $n \geq 3$, $\frac{n^n}{4} > n!$

Remember:

- n^n is n multiplied by itself n times. So, when $n = 0$, $n^n = 1$; when $n > 0$, $n^n = n * (n^{n-1})$.
- $n!$ is $n * (n-1) * (n-2) * \cdots * 3 * 2 * 1$. So, when $n = 0$, $n! = 1$; when $n > 0$, $n! = n * (n-1)!$

Assume the very useful fact that $k^{k-1} \geq (k-1)^{k-1}$ for integers $k \geq 1$.
Here are all the values smaller than 100 of each of the functions. ($\frac{5^5}{4}$ and 5! are both larger than 100.)

n	0	1	2	3	4
$\frac{n^n}{4}$	0.25	0.25	1	6.75	64
$n!$	1	1	2	6	24

Complete the following proof of the theorem by filling in the areas marked **TODO** on the right side of the page. Remember to briefly justify any step you take that is not a basic arithmetic step.

SOLUTIONS ARE INLINE BELOW.

Proof by induction:
Base Case: When $n = 3$, $\frac{3^3}{4} = 6.75$ which is greater than 6, which is 3!.
Inductive step: Consider an arbitrary integer (or whole/natural number) $n > 3$.

SOLUTION NOTE: $n \geq 4$ means the same thing here. Note also that the correct inequality here (both whether it's ≥ 4 or ≥ 3 and whether it's n or some other variable) depends on what you assume in your IH and say you're going to prove in your IS below. Rule of thumb: for the smallest legal value you allow your variable to take on, your IH should be about the $n = 3$ (base) case or, similarly, your IS should be about the first value larger than the base case, i.e., $n = 4$.

Induction Hypothesis: Assume that $\frac{(n-1)^{n-1}}{4} > (n-1)!$
Now, we show that $\frac{n^n}{4} > n!$:

$$\begin{aligned}
\frac{n^n}{4} &= n * \frac{n^{n-1}}{4} && \text{by the definition of } n^n \text{ above since } n > 0 \text{ (but no just. necessary)} \\
&\geq n * \frac{(n-1)^{n-1}}{4} && \text{by the assumption given since } n \geq 1 \text{ (just. required)} \\
&> n * (n-1)! && \text{by the IH (just. required)} \\
&= n! && \text{by the definition of } n! \text{ above since } n > 0 \text{ (but no just. necessary)}
\end{aligned}$$

Therefore, by the principle of mathematical induction, for integers $n \geq 3$, $\frac{n^n}{4} > n!$

SOLUTION NOTE: Even if you had no idea how to proceed with the algebra of the final argument, you can get quite a bit of the way there from the recursive structures given.

First, the recursive structures suggest a base case of $n = 0$ and inductive case of $n > 0$ because each one is divided into two cases when $n = 0$ and when $n > 0$, and the $n > 0$ cases have a recursive appearence of the structure being defined (e.g., $(n-1)!$ is a recursive appearence of factorial). However, since we're proving this for $n \geq 3$, it makes more sense to redefine these for a base case when $n = 3$ (e.g., $n! = 3! = 6$ when $n = 3$) and an inductive case for $n > 3$. That part's a bit tricky!

Next, in the inductive step, we know we want to use the inductive cases of the two definitions we were given. We also know we're proving an inequality; so, we'll start from one side and work to the other.

Right away, you should know that you want to change n^n into $n * (n^{n-1})$ working from the left side and $n!$ to $n * (n-1)!$ working from the right. That's given in the recursive structures!

Next, once you see $(n-1)!$, that's your "recursive appearence". The whole purpose of your induction hypothesis is to act on the recursive appearence that comes out of your recursive structure. So, as soon as you see it, you should use your IH. In this case, that means replacing $(n-1)!$ using the fact that $(n-1)! < \frac{(n-1)^{(n-1)}}{4}$.

You might also be able to tell that you need to use the assumption we gave.

The last little bit is pure algebra (and is worth a tiny fraction of the marking scheme).

The moral of that story is that even if you think you are "bad at math", you should practice the structural pieces of induction so you can get as far as possible in an induction proof without needing the "math". Train yourself to succeed! :)

3 Predicate Logic to Proof Structure [6 marks]

Consider the following theorem:

$$(\exists p \in T, A(p,p)) \rightarrow \forall q \in S, \exists r \in T, \exists s \in T, A(r,s) \wedge (\sim B(q,r) \vee \sim B(q,s))$$

There is **not enough information** about A, B, S, and T to prove this theorem. Instead, lay out as much of the structure of a direct proof of this theorem as you can. **Whenever you choose a specific value for a variable, specify what (if anything) this choice can depend on and any important constraints (limitations) on that variable.**

Please list each distinct step after its own bullet point so it's easy for us to follow your proof structure! (Our solution has six bullet points, as shown.)

Proof/SOLUTION:

- Assume $\exists p \in T, A(p,p)$. (Note: it is **incorrect** to then try to prove $\exists p \in T, A(p,p)$. This is an assumption, not a theorem to prove!)
- WLOG, let q be an arbitrary element of S
- Choose $r =$ _____ (a value in T that can depend on p and q)
- Choose $s =$ _____ (a value in T that can depend on p and q; OK to say (but not useful/insightful) that it can depend on r)
- Show $A(r,s)$
- Show $\sim B(q,r) \vee \sim B(q,s)$

Reasonable "extra" steps on that last line are fine if stated clearly and correctly—e.g., that you may want to prove one or the other of the two disjuncts (sides of the "or")—but not necessary.

How do we know all these steps? We repeatedly look at what the outermost operator in the statement and then use the proof strategies sheet to select the most straightforward strategy to strip off that operator. That leaves us with a new outermost operator to work with.

We don't know which values of r and s to choose without more information about the domains and predicates in this problem.

4 Proof by Contrapositive [6 marks]

Prove the following theorem using as a central step a **proof by contrapositive**:

> For any integer x, if $3x^2 + 2x - 7$ is odd, then x is even.

Be sure that you clearly and appropriately address each piece of the theorem in your proof. Substantial partial credit is available for a correctly structured (but incomplete) proof.

SOLUTION: We write this as bullet-points to make the structure clear (but we wouldn't normally do so). Note that the contrapositive of "if $3x^2 + 2x - 7$ is odd, then x is even" is "if x is odd, then $3x^2 + 2x - 7$ is even".

- WLOG, let x be an arbitrary integer.
- Assume that x is odd, i.e., $x = 2k + 1$ for some (unknown) integer k
- We must now show that $3x^2+2x-7$ is even, i.e., that there is an integer k' such that $3x^2+2x-7 = 2k'$:
  - Choose $k' = 6k^2 + 8k - 1$, which is an integer, since k, 6, 8, and 1 are integers, and the integers are closed under addition, subtraction, and multiplication.
  - $3x^2 + 2x - 7 = 3(2k + 1)^2 + 2(2k + 1) - 7 = 12k^2 + 16k - 2 = 2(6k^2 + 8k - 1) = 2k'$

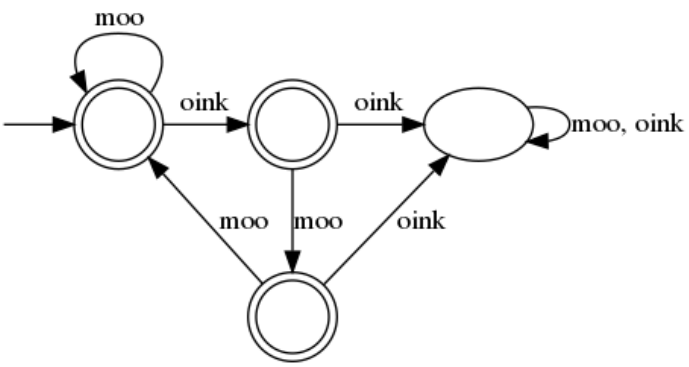
QED

How did we know what to make k' equal to? We left it blank, worked through the rest of the proof up to $2(6k^2 + 8k - 1)$ (knowing that we wanted to reach something of the form $2 *$ some integer), and then went back and filled in the blank!

(It's possible to use modular arithmetic to prove this, although we expect a bit of justification of steps, e.g., that an even times an odd is an even or an odd times an odd is an odd. However, we recommend being comfortable with the approach above!)

5 DFAs and Regexes [7 marks]

1. The following DFA accepts a certain pattern of moos and oinks:



Describe the pattern accepted by this DFA, using plain English, in no more than two sentences. **[3 marks]**

SOLUTION: Any two oinks are separated by at least two moos.

(It's not quite right to say that any oink is directly followed by at least two moos because the **last** oink needn't be. You could say "any oink except the last is directly followed by at least two moos. The last oink (if any) is followed by zero or more moos." This is a much more cumbersome statement but actually translates quite nicely into a regular expression.)

2. Write a regular expression that accepts the same sequences of oinks and moos as the DFA. Use **O** to represent an oink, and **M** to represent a moo. Otherwise, use only the regular expression language provided at the bottom of page 3 of the reference sheet. **[3 marks]**

   SOLUTION: `(M|OMM)*(OM*)?`

   How did we get this? This says that the "meat" (so to speak) of the regular expression is a bunch of moos or oinks-followed-by-two-moos but that the last oink (if any) is followed by zero or more moos. Compare that to our "cumbersome" English above.

   Another way to see this is that `(M|OMM)` are two patterns that "loop back" to the start state. We can follow that loop as often as we like. Then, the string must end by getting us to one of the accepting states.

   (Because it makes the "parsing" of any string unambiguous, I actually slightly prefer this also-correct expression `(M|OMM)*(OM?)?`.)

3. Moos and oinks are animal sounds and not numbers or letters. Given this, would it be possible to translate this DFA to a circuit? **[1 mark]**

   SOLUTION: POSSIBLE

   Of course! All term long, we have been using bits to represent all kinds of things. We can just choose, e.g., to have 1 mean "moo" and 0 mean "oink" (or vice versa or some more bizarre solution), at which point nothing special is happening here. This is also a lesson that CPSC 110 works hard to impart: you can use your representation tools to represent any domain you like. (Well, actually, CPSC 121 explores one or two limits to what's possible, but they're far beyond the scope of this question!)

6 Extending the Working Computer [6 marks]

EVERYTHING on this page down to the words "QUESTION BEGINS HERE" is identical to the similar tutorial/assignment problem.

A friend suggests that we should have a "subtract a constant" (or "immediate" in working computer terms) instruction. They propose that the constant can be 1 byte long or 2 bytes long and that the instruction includes one register from which we subtract the constant.

So, for example:

```
subi %ebx, $7
```

would subtract 7 from the value in register `%ebx` and update `%ebx` with the resulting value.

Your friend suggests a `subi` to subtract a 2-byte constant would look like:

C	0	rA	F	V	V	V	V

where C is a hexadecimal number (12 in decimal or binary 1100) and is the opcode for the new instruction. 0 is hexadecimal 0 and is the function code, which is ignored for this instruction, as for most instructions. rA is the register to subtract a constant from. So, if we wanted `%ebx`, it would be a 3. F is a hexadecimal number (decimal 15 or binary 1111) indicating that we do not want a second register. The 4 V's are space

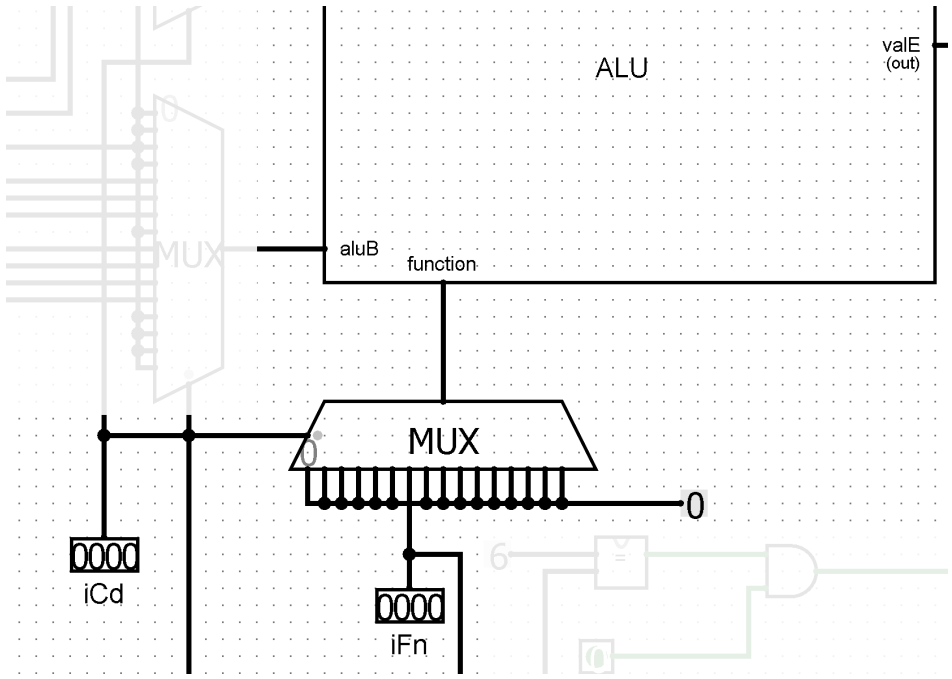
for the 4 hexadecimal digits representing the 2-byte long constant value to subtract, interpreted as a signed binary number.

The friend says the 1-byte version should look like:

C	0	rA	F	V	V

QUESTION BEGINS HERE

Now, consider the following portion of the "execute" stage of the computer, where the unfaded parts focus on computing the ALU's *function* (the operation the ALU is to perform). Note: when the function is 0, the ALU performs addition; when the function is 1, the ALU performs subtraction; for other functions, it performs other operations.



The input `iCd` in this image is the opcode (the first hexadecimal digit) of the current instruction. The input `iFn` is the function code (the second hexadecimal digit) of the current instruction.

1. We plan to support **both versions** of the `subi` instruction by keeping the opcode of the 2-byte version of `subi` as C but changing the opcode of the 1-byte version of `subi` from C to D.

   Use the image above to explain how you would change this portion of the circuit to implement this plan. [3 marks]

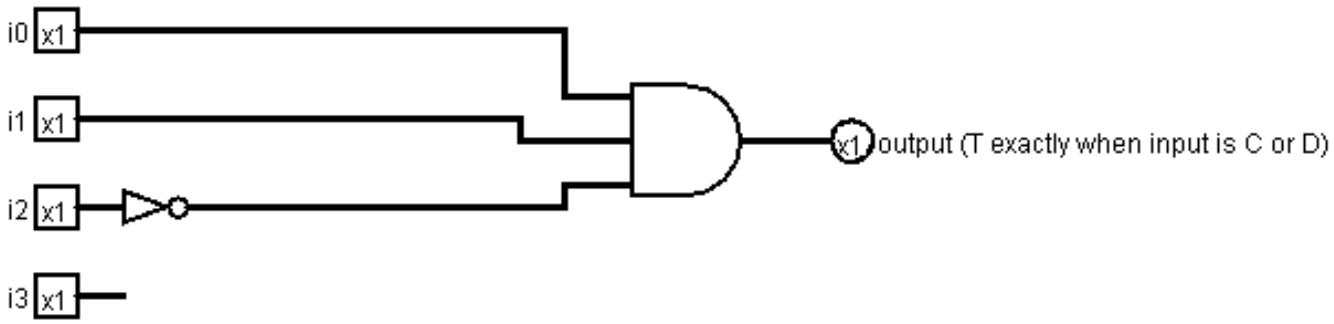
   SOLUTION: We just describe it here, although some circles in the diagram are **super** helpful in your answer! The wires leading into the unfaded MUX numbered 12 and 13 (C and D, counting from 0 on the left hand side, i.e., the 3rd and 4th from the right) must be changed to take in 1 (meaning subtraction) rather than 0 as they take in now.

   (An alternate solution that is more complex but perhaps a better design is to change the instructions above so that their second 4 bits are 1's rather than 0's. Then, make the same change discussed in the previous paragraph except wire inputs 12 and 13 to the `iFn` input rather than to 1. The advantage of this approach is that it means you can easily design `addi` and other arithmetic instructions on immediates/constants just by letting the function code take on any of its legal values.)

2. **Design a circuit** that takes as input the 4-bit `iCd` $i_0 i_1 i_2 i_3$ and produces true as output if the `iCd` is C or D and false otherwise (i.e., determines if the opcode is for one of the `subi` instructions). Use **only** inverters (NOTs) and 2-, 3-, or 4-input AND and OR gates in your circuit. **[3 marks]**

   (Note: when the `iCd` is C, $i_0 i_1 i_2 i_3 = 1100$.)

   SOLUTION:



   There's also a natural DNF solution that's basically the one we gave AND'd with i_3 in one case and $\sim i_3$ in the other and then OR'd together. This is the one you'd come up with if you followed the mechanical process we learned to translate a truth table into a logical expression and then a circuit. It's fine and correct if a bit more time-consuming to draw.

7 Induction on Algorithms [6 marks]

The following procedure (i.e., algorithm or function) adds the numbers in a list named A in a strange way as part of an efficient parallel sorting algorithm. Its second parameter c is a positive integer. It calls a helper function HELPER that works correctly, but for which we do not provide code here.

procedure SPLITANDSUM(A, c)
  if A has c or fewer elements **then**
    return HELPER(A)
  else
    Let n be the length of A
    Let A_l be the left half of A, i.e., elements with indexes i such that $0 \leq i < \lfloor \frac{n}{2} \rfloor$
    Let A_r be the right half of A, i.e., elements with indexes i such that $\lfloor \frac{n}{2} \rfloor \leq i < n$
    return SPLITANDSUM(A_l, c) + SPLITANDSUM(A_r, c)
  end if
end procedure

Imagine that we want to prove some new property $P(n)$ that describes how SPLITANDSUM operates on a list of length n. We have **already separately proven** that $P(n)$ is true for all natural numbers $n \geq 0$ for calls to HELPER.

Give the best answers possible to the following questions about a proof by induction that $P(n)$ holds for all natural numbers $n \geq 0$ when calling SPLITANDSUM, **without knowing what P is**.

1. For what value(s) of n will we write base case(s)? **[2 marks]**

   SOLUTION: $n \leq c$. That is, the same as the base case in the recursive algorithm! (Note that $c \geq 1$, which is important since otherwise we might not have 1 as a base case. If we don't, then $\lceil \frac{n}{2} \rceil = n$ when $n = 1$, which means our induction proof has an infinite chain ("loop" or "recursion") and is incorrect.

2. How many times will we invoke the induction hypothesis (IH) in the proof? **[1 mark]**

   SOLUTION: Twice. (Once for each recursive call in the algorithm.)

   How do we know? Again, look at the algorithm. In its recursive case, there are two recursive appearences of the function; so, we'll invoke the IH twice.

3. Assuming that in the inductive step we want to establish $P(n)$ for $n = k$ (where k is an appropriate natural number), for what specific value(s) of n will we invoke the IH in the proof? **[2 marks]**

   SOLUTION: For $n = \lfloor \frac{k}{2} \rfloor$ (the length of the left half of A, because it is the parameter in the first recursive call) and for $n = k - \lfloor \frac{k}{2} \rfloor = \lceil \frac{k}{2} \rceil$ (the length of the other half, because it is the parameter in the second recursive call). Note that either version of the latter is fine as far as we are concerned.

   Note that we've again just read our answer off of the algorithm (the recursive structure over which we're performing induction)!

4. For this algorithm to work correctly, what should the function Helper do with A? **[1 mark]**

   SOLUTION: Sum the elements in the list A.

   How it does that is irrelevant, but it does need to accomplish that or else SplitAndSum won't do the right thing in **its** base case.