Practice test for Pearl 000 of Computer Science (202001021)

9 September 2021

Answers

---

1. Binary numbers

(a) 101010111100
Recall that conversion from hex to binary can be done easily digit by digit: A = 10 in decimal = 1010 in binary, etc.

(b) –11
Recall that 2-complement is a positional notation; so the first 1 has a weight of -2^4, the next 1 has a weight of 2^2, and the last one has a weight of 2^0, making for a total of $-16 + 4 + 1 = -11$.

(c) This doubles the number. This is because each bit moves to a position where its weight is twice as large. E.g., the right-most bit (least significant) moves from a position with weight 2^0 to a position with weight 2^1, etc. Thus, we double all terms, which results in doubling the sum as well.

---

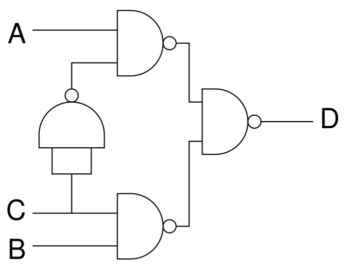
2. Boolean logic

(a)

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(b) We apply DeMorgan's theorem: $\overline{C} \cdot A + C \cdot B = \overline{\overline{(\overline{C} \cdot A)} \cdot \overline{(C \cdot B)}}$
The parentheses are not really needed here, as the bar above already makes clear which parts belong together.

(c) Drawing this is straightforward, as the expression is already in the form of (N)AND operations.



Note the use of one 2-input NAND gate as an inverter to get the $\overline{C}$ signal.

3. A processor

	read address 1 / write address	read address 2	instruction
Timeslot 0	2	1	0
Timeslot 1	1	2	1
Timeslot 2	3	1	0
Timeslot 3			
Timeslot 4			
...			

The first instruction adds R1 by R2 and stores the result in R2.
The second instruction multiplies R1 by R2 (which by now contains the sum of the original contents of R1 and R2), and stores the result in R1.
The third instruction adds R1 (containing the result of our calculation) to R3, and stores the result in R3. Since it was given that R3 contained 0, this effectively copies the content of R1 into R3.

Note that the first instruction was chosen such that it writes into R2, not into R1, because we still need the (original) contents of R1 in the next step.

In this example, I wrote the read and write addresses in decimal, but of course writing them in binary is also good.

4. An AVR program

Best is to make a table in which you write down after every instruction what the contents of the registers are:

R16	R17	R18	R19	explanation
\$0				
\$0	\$80			
\$0	\$80	\$3		See remark.
	\$80			
\$1				
			\$1	
			–2	result from $1 - 3 = -2$; in hexadecimal it would be \$FE, as can be seen by first converting to an 8-bit 2-complement binary number, and converting that to hexadecimal
				branch is performed, because previous result is not 0, back to the ADD instruction
	\$81			
\$2				
			\$2	
			–1	
				branch is performed
	\$83			
\$3				
			\$3	
			0	
				branch is not performed, because previous result is 0

Remark: in the first few lines, I indicated in small font also the contents of registers that that instruction did not write into. Of course, this gets boring very quickly, and doesn't add much clarity, so I stopped doing it in the rest of the table.

The question asks for the final contents of R16, R17, R18 and 19 in decimal; those are 3, 131, 3 and 0, respectively.

In the table you can count that 15 “normal” instructions were performed, each taking 1 cycle; and three branch instructions, two of which resulted in a jump, taking 2 cycles each, and one that did not, taking 1 cycle; so total is $15 + 2 \times 2 + 1 = 20$ cycles.

---

5. Problem 5

This program calculates 2XY.

To see this, first note that there are two loops, each encompassing and ADD, a DEC, and a BRNE instruction.

The first of these repeatedly adds R17 to R20. R20 starts at 0 (thanks to the 4th LDI), and R17 is the number X. This loop is executed Y times: that's because we start with R18 equal to Y, and everytime we go through the loop, R18 gets decremented by 1, and we stop looping around when R18 becomes zero (thanks to the BRNE, Branch if not zero). Thus, after this loop, we've added X to R20, Y times, thus R20 equals XY.

The second loop does essentially the same thing: it repeatedly adds Y (now stored in R19, thanks to the MOV instruction) to R20, and it does this X times (namely until R17 becomes 0, while R17 started out as X.

Thus, the final result is that R20 equals 2XY.

Of course, this way of calculating 2XY is unnecessarily complicated and slow. It's just an exercise here, which challenges you to imagine what the final result of all the detailed steps of the program is.