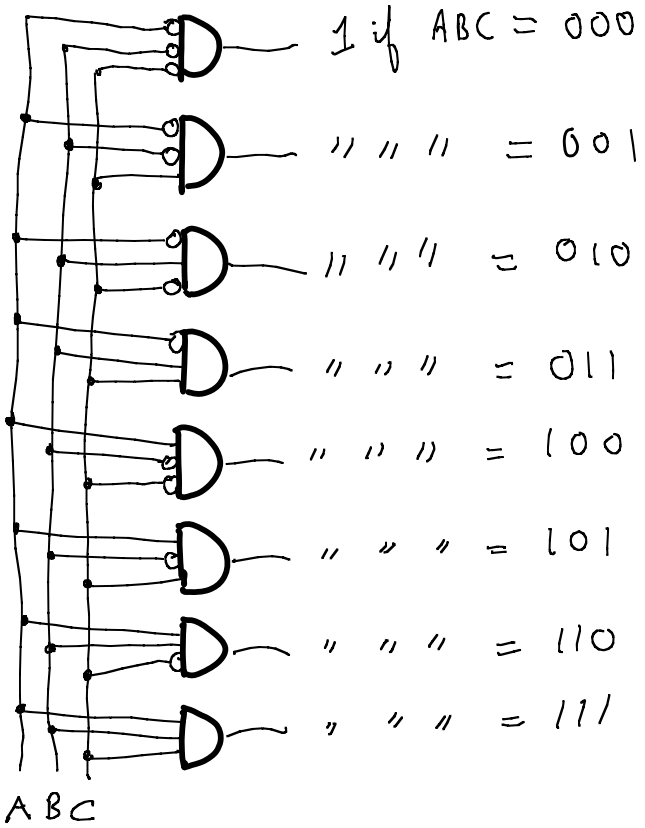
Reading: PP-chp 3:

3.3 (decoder, mux, FA, PLA)
3.4 (R-S latch, register)
3.5 (memory)
3.6 (sequential machines, FSM)
3.7 (LC-3 datapath)

§ Problems, PP-chp 3:
§ 3.12 3-Dec, show minterm exp.
§ 3.13 5-dec, show num output lines.
§ 3.14 16X1 mux, how many select lines?
§ 3.19 explain mux ckt, s-r latch
§ 3.20 truth table => trans. Ckt -(-a.b)
§ 3.22 4X1 mux, from 2x1 MUXs
§ 3.24a 2x1 mux, identify input (select)
§ 3.25 gate delays for 32-Add
§ 3.27, simplified latch, figure out
§ 3.29 d-latch transparency
§ 3.30a 2-in-3-out comparator truth table.
§ 3.30b, logic ckt for (a).
§ 3.30c, 4-bit EQUAL from 1-bit 2X3 comparators
§ 3.31 #word X word_size = mem. Size
§ 3.32, addressability vs. address
§ 3.33a row X col addressing: find 4-th word
§ 3.33b #selects for 60 words?
§ 3.33c #words max for #select=3?
§ 3.43a fsm truth table
§ 3.43b state diagram for (a)

3.12 | 3-input decoder



3.13 | 5-input decoder has output lines for each of,

ABCDE = 00000
ABCDE = 00001
ABCDE = 00010
⋮ ⋮
ABCDE = 11111

$11111_2 = 100000 - 1$

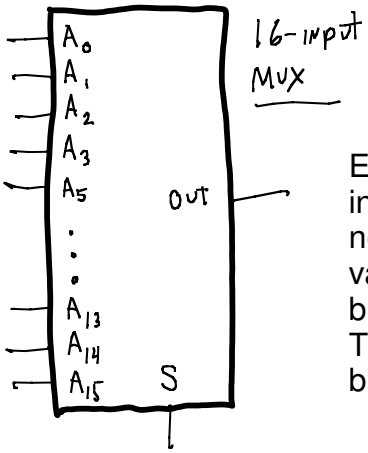
$= 2^5 - 1$

$= 32 - 1$

$= 31$

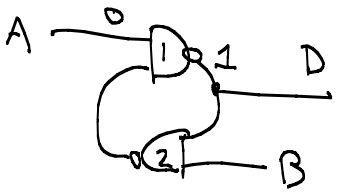
Since binary 11111 represents 31, there are 32 bit patterns from 00000 to 11111. Each pattern activates a separate output signal. Therefore, the decoder has 32 outputs.

3.14 |



Every MUX has a single output, which is fed from whichever input is selected by the "S" select signal. This 16-input MUX needs a select signal that can take on any of 16 different values, one for each input. Since 1111 is binary for 15, the binary numbers 0000 to 1111 are 16 different 4-bit numbers. This is sufficient to control the selection of a 16-input MUX: 4 bits are needed for the "S" signal.

3.19 |



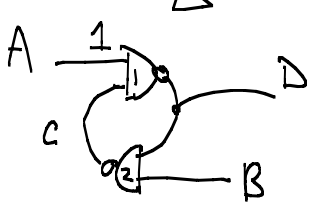
(D should be labeled "Q")

D = 1 regardless of B

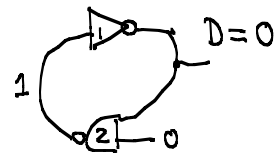
ABC	(MUX) D	
000	0	follow C
001	1	
010	0	
011	1	
100	0	follow B
101	0	
110	1	
111	1	

AB	NAND
00	1
01	1
10	1
11	0

A → 1

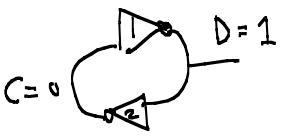


Case B = 0



(1) acts as invert
(2) outputs 1

Case B = 1



(1) + (2) act as invert

AB	D (Latch)
00	1
01	1
10	0
11	??

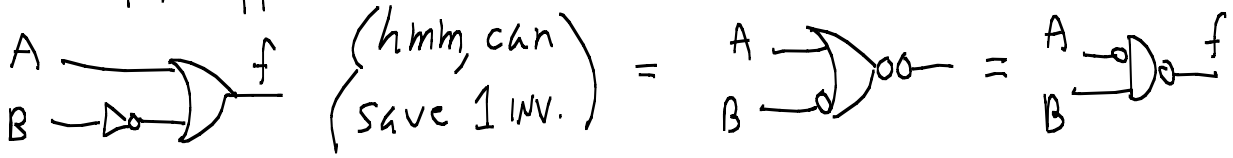
(3.19, cont.)
The thing to notice about the two truth tables above is that for the latch we cannot determine the value of D from the values of (A, B) = (1, 1). The reason is that if the signal sequence for (A, B) had been (1, 1), (0, 1), (1, 1), then D = 1. But if the sequence had been (1, 1), (1, 0), (1, 1), then D = 0. That is, the state of the latch remembers prior inputs. That is not true in the case of the MUX as its output is completely specified by the current inputs. Here's an interesting question about the latch: What if the input sequence had been
(1, 1), (0, 0), (1, 1), what would the value of D be? Hint: suppose instead you have a cycle built from two inverters, and both inputs are temporarily grounded, then disconnected from ground at the same instant. What effect does the finite rise time of an inverter's output have on the state of the system?

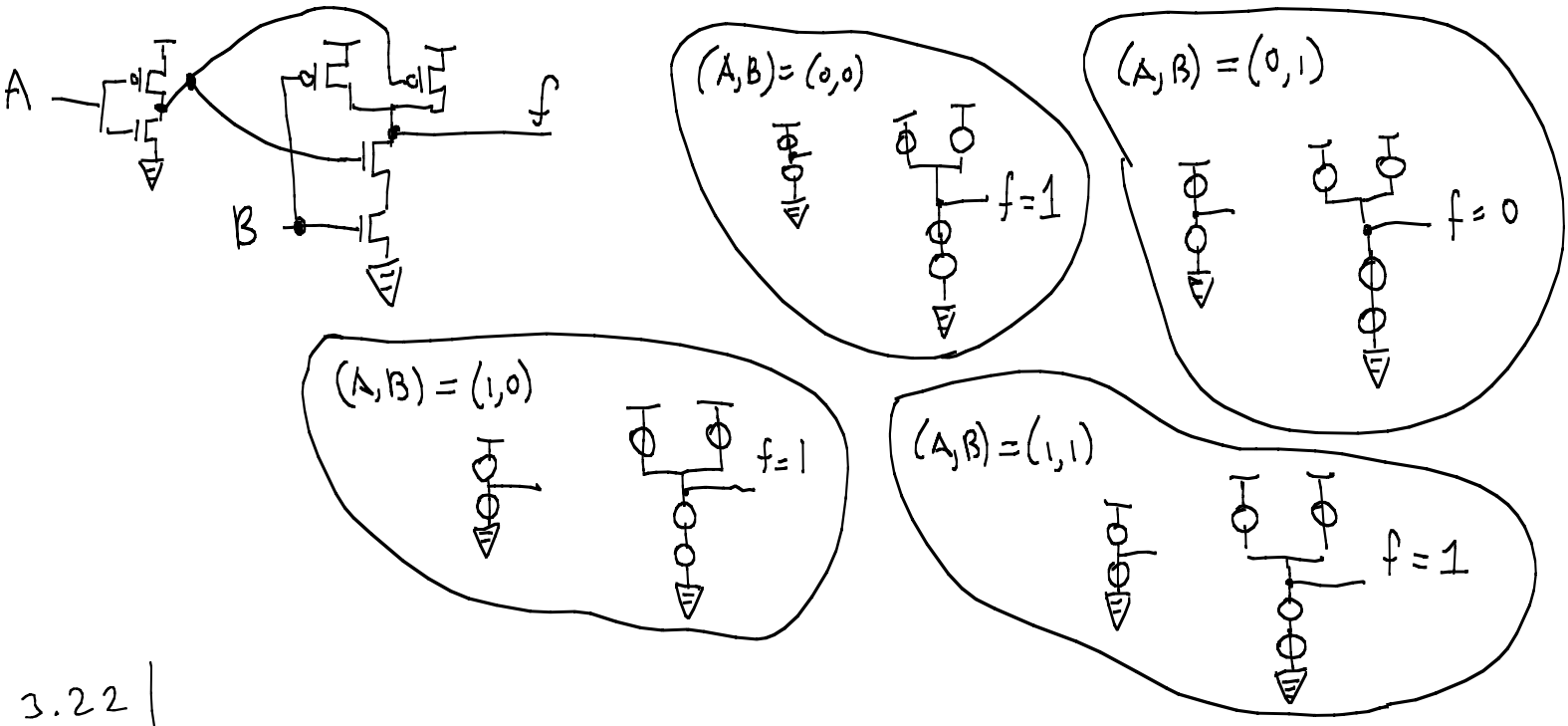
3.20

A	B	f
0	0	1
0	1	0
1	0	0
1	1	1

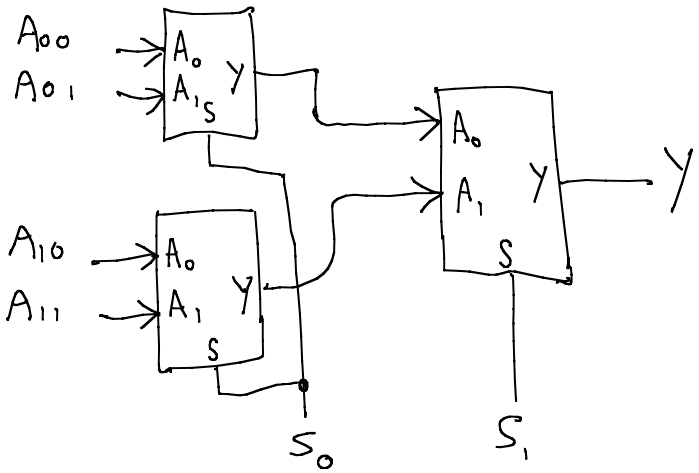
this calls for a max-term expansion.

$\bar{f} = \bar{A}B \qquad f = \bar{\bar{f}} = \overline{\bar{A}B} = A + \bar{B}$





3.22



Truth table, 4X1 MUX

S1	S0	Y
0	0	A00
0	1	A01
1	0	A10
1	1	A11

Truth table, 2X1 MUX

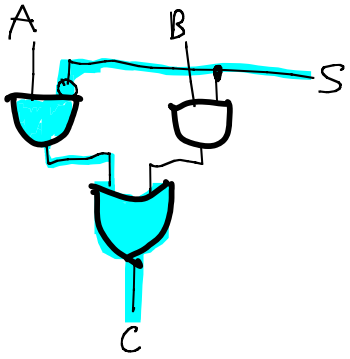
S	Y
0	A0
1	A1

3.24a

Signal X selects whether to add word B = (B2, B1, B0) or add word C = (C2, C1, C0) to word A = (A2, A1, A0).

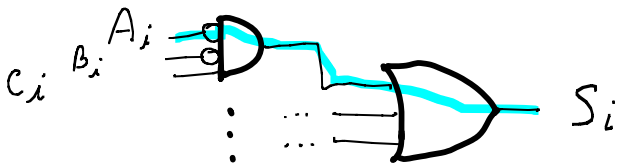
3.25

(a) (fig 3.12)



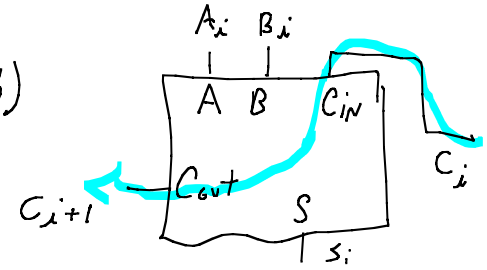
Delay for S = 0 is 3 : NOT, AND, OR

(b) (fig 3.15)



This example shows 3 delays for the sum bit. The carry bit requires the same delay.

(c) (fig 3.16)

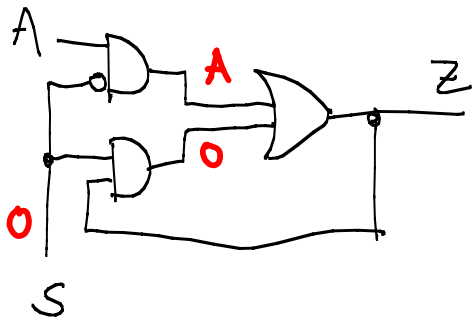


Each 1-bit adder requires 3 gate delays for the Ci signal to propagate to the Ci+1 output. For the first stage, getting C1 does not depend on C0 because C0 = 0 always, but C1 also depends on A1 and B1 which takes 3 delays. Since there are 4 1-bit adders chained together, Ci to Ci+1, the total delay is 4*(3) = 12 delays.

(d) Chaining together 8 of the 4-bit adders from (c) would give us a 32-bit adder with $8 \cdot 12 = 96$ gate delays.

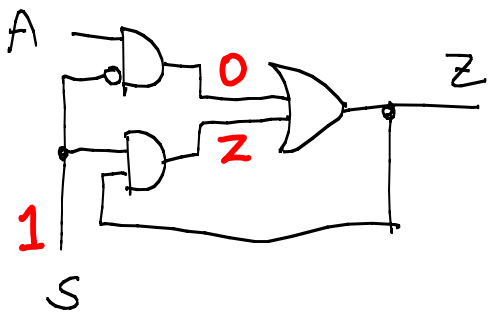
3.27

(a)



for S=0, Z=A.

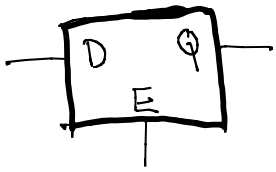
(b)

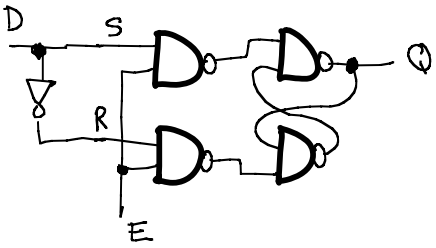


for S=1, Z=Z.

(c) Yes, this is a latch. When S = 0, the latch is transparent with Z = A, whatever A is (0 or 1), Z simply follows A. But, when S = 1, The A input is ignored, and the output remains Z, whatever Z was before S changed to 1.

3.29

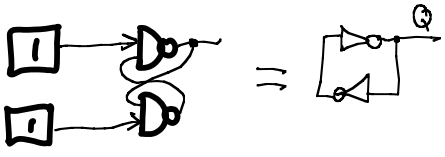




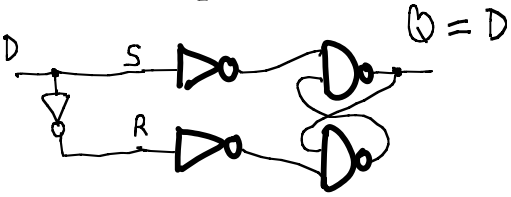
Here's a D-latch. Suppose Q = 1 and E = 0. The D input can change between 0 and 1 many times, but Q will remain 1. Suppose D goes to 0, and then E goes to 1. Then Q = 0. There is now no way to tell that Q was 1 before. If E now goes to 0, i.e., our D-latch is written into, there will be no information about its prior state.

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

E = 0

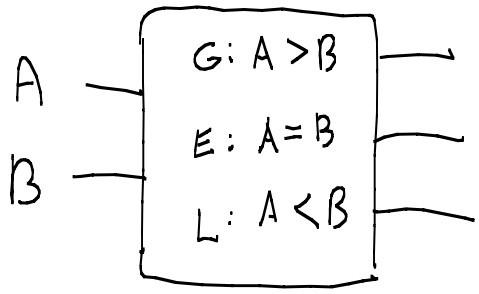


E = 1



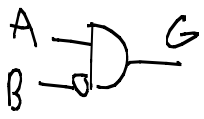
3.30

(a)

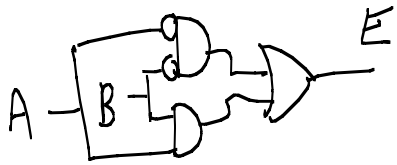


A	B	G	E	L
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

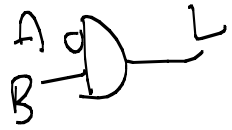
(b) $G = A\bar{B} \Rightarrow$



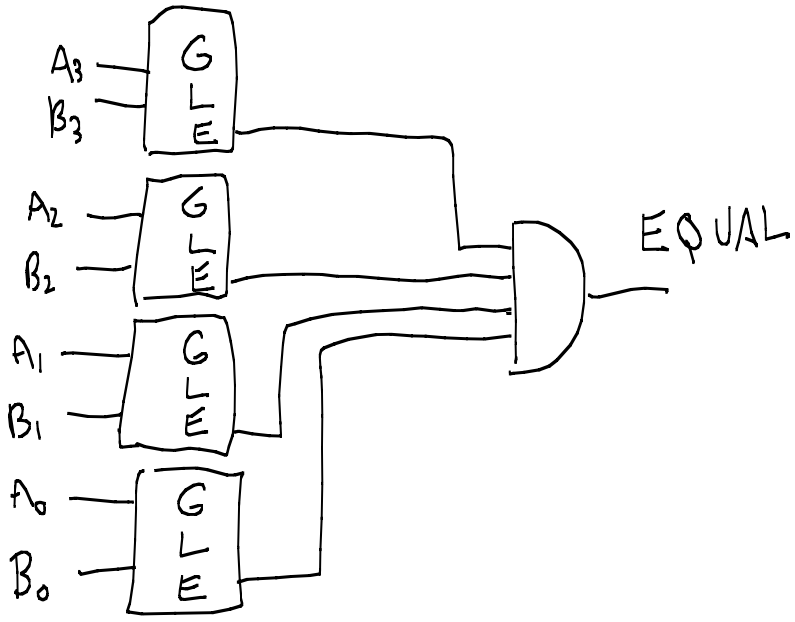
$E = \bar{A}\bar{B} + AB \Rightarrow$



$L = \bar{A}B \longrightarrow$

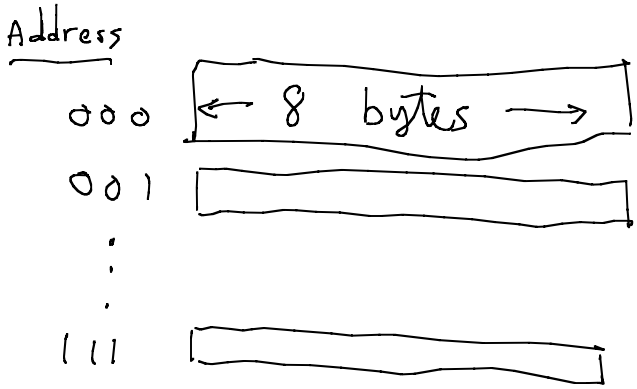


(c)



if $A = B$, then for every bit, $A_i = B_i$

3.31



8 word x 8 bytes per word = 64 B memory

3.32

Addressability is the size of the memory's data units; that is, the number of wires coming out of the memory's MUX, which is the same as the number of wires going into the memory's data input. However, memories are often packaged together to form a single unit that delivers word-sized (or larger) data. For instance, a machine's word size might be 4 bytes, the memory unit delivers 4-byte words by accessing 4 individual memories, each delivering 1 byte. The addressability can also be thought of as the number of bits one advances in the memory when the lowest address bit is changed.

A memory address is the input to the individual memory's MUX and DEMUX select lines. The address specifies which memory cell to access, the addressability specifies what the granularity is in addressing the memory. So, with byte-addressability, a 32-bit word sized machine will access a 4-byte word at each byte address. But the 4-byte word starting at the next byte address overlaps the previous word by 3 bytes.

(3.33a)

We want to access the 4th word, which is the bottom row in Figure 3.21. The address decoder will set that row's decoder output to 1 if the address bits are A[1:0] = (1, 1). That will in turn gate each latch's output to the OR gates for D[2], D[1], and D[0], and thereby produce the data at the memory's output. We do not want to disturb the data in that row. Since the decoder's output is ANDed with the WE signal, and the output of the AND feeds into each latch's ENABLE input, we must have the WE signal be 0 so that the latches will not have their contents erased and replaced with the bits on the data input lines, Di [2:0], which would also be the output from the memory wires D[2:0].

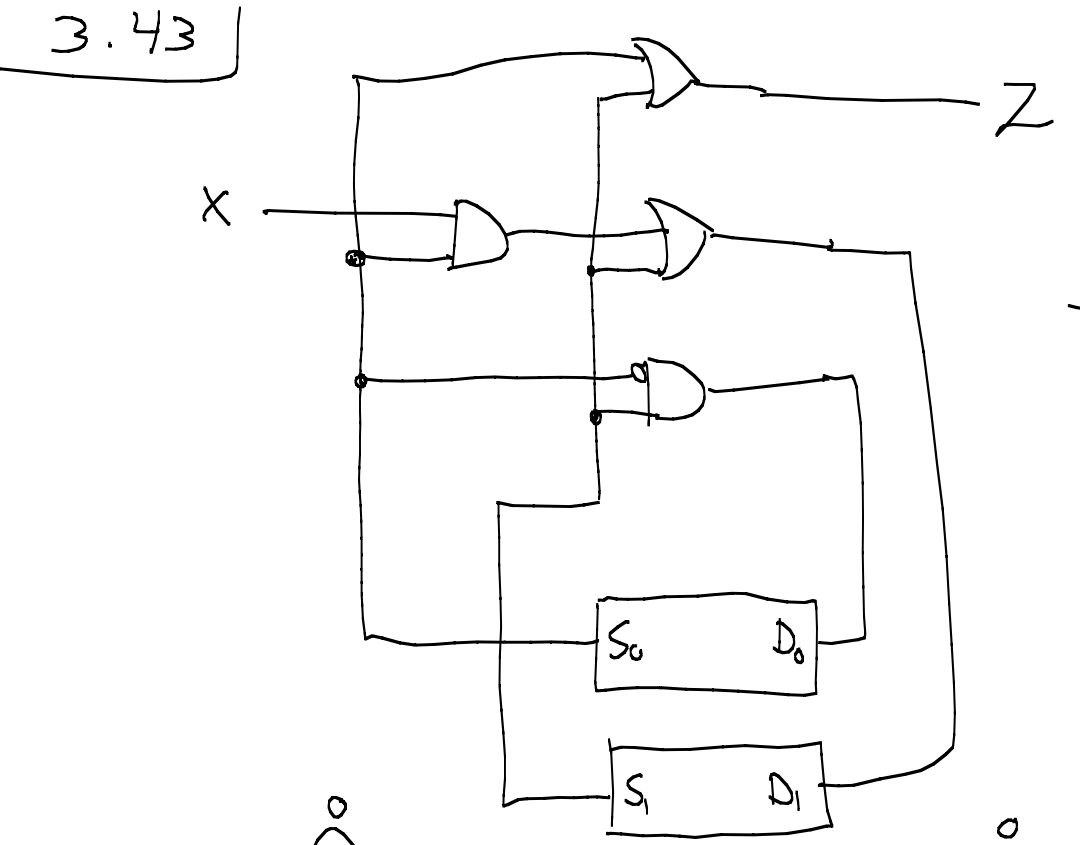
(3.33b)

We need k address lines so that we can address any of the 60 words of memory. The address lines feed into the select lines of the memory's MUX. A MUX with k select lines has 2^k input lines. So, we need to find a k such that 2^k is at least 60. If we try k = 5, we can only address 32 words, but with k = 6 we can address 64 words. 64 is more than enough; so 6 address bit will work. We did not change the size of the memory's word; so, the addressability is still 3-bits.

(3.33c)

The PC needs to address the entire memory. From part (b), that is 60 words. We need at least 6 bits of address to access 60 words, so the PC must be at least 6 bits. However, the PC can address from 000000 to 111111. This last number is (1000000 - 1), which is 2**6 -1, which is 64 - 1 = 63. So, the total number of different addresses the PC can hold is 64. Thus, we could add 4 words to the memory and still be able to have the PC hold the address of any memory word.

3.43



S_1	S_0	X	D_1	D_0	Z
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	1

(b)