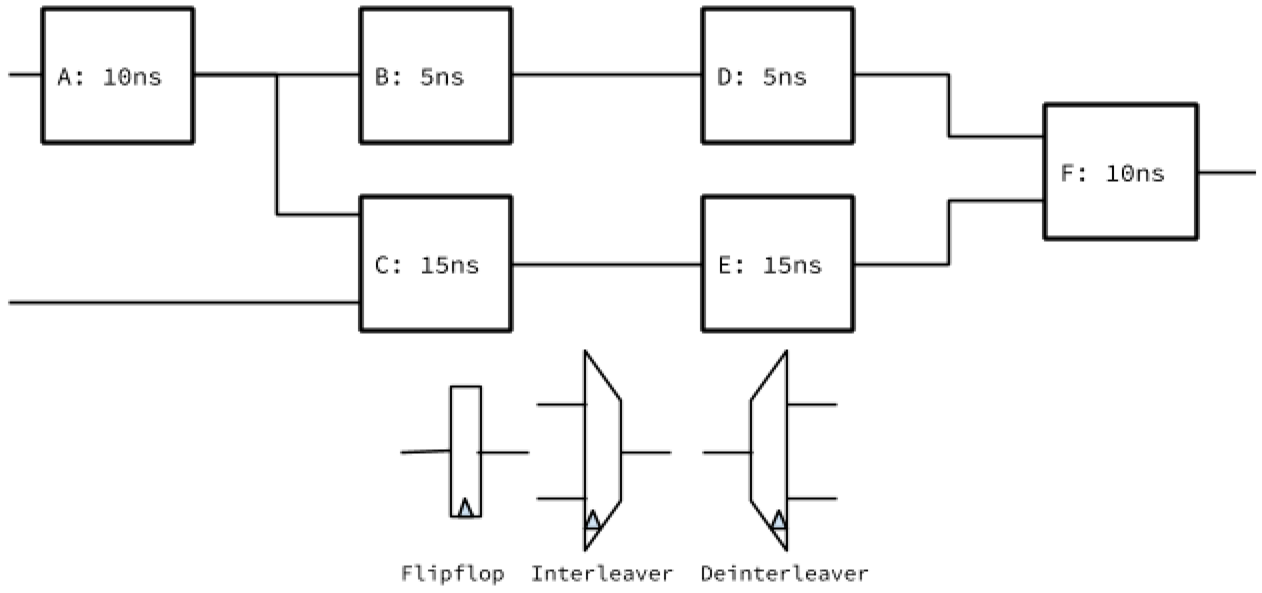
Department of Electrical and Computer Engineering
The University of Texas at Austin
EE 460N/382N.1, Spring 2018
Instructor: Dr. Mattan Erez
TAs: Steven Flolid, Armand Behroozi, Abraham Gonzalez

Problem Set 3 Solutions

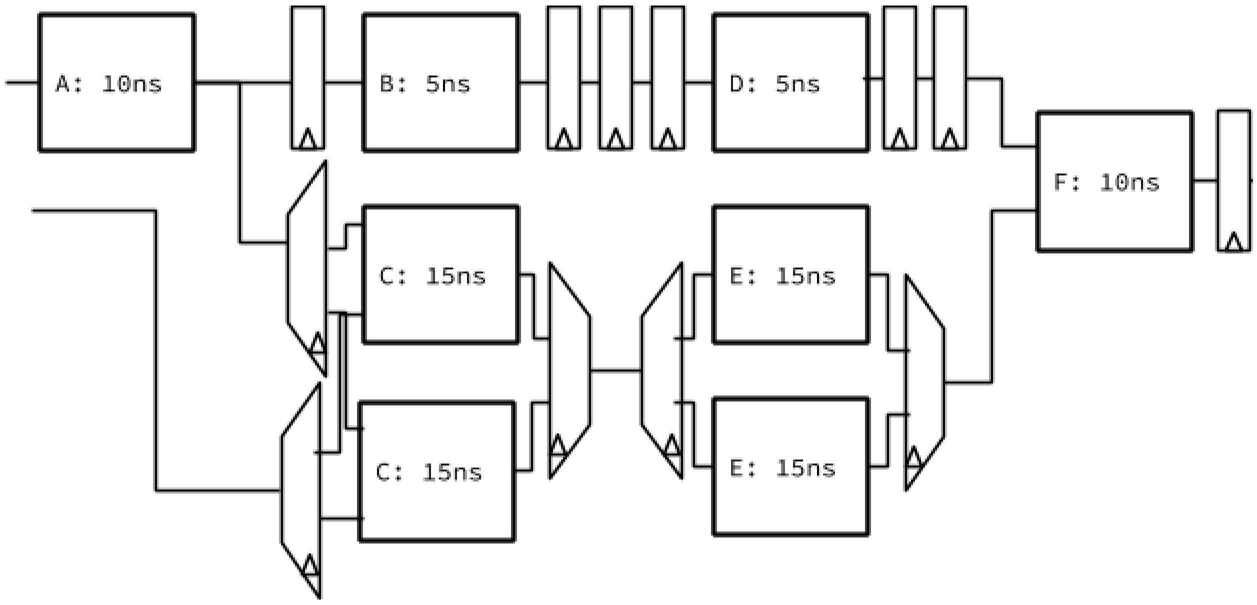
1. An old LC-3b state diagram contained errors in states 4, 20, and 21. We have posted both versions of the handout: wrong and corrected. Briefly explain the problem we have corrected.

In the wrong version, instructions JSR R7 and JSRR R7 would execute incorrectly. Their base register (R7) would be overwritten with the value of the PC in state 4. Thus, the addresses generated in states 20 and 21 would come out wrong.

2. Pipeline the circuit below. Optimize throughput (inputs processed per ns) and cost ($). You may add any number of the blocks in the circuit (A-F), edge-triggered flip-flops, edge-triggered interleavers, and edge-triggered de-interleavers (see figure below). No other modifications are permitted. The latency of flip-flops, interleavers, and de-interleavers is 0ns. The cost of any item (whether already there or one you add), regardless of which item is $1. You have a total budget of $25 and the components already drawn below cost $6. Draw your answer on top of the diagram below if at all possible. What is the throughput, latency and cost of your solution?
Prior Test Question (Fall 2017)



There is possibly more than a single correct solution. Things to think about: (a) pipeline needs to be well-formed with (i) same number of latches on all paths, and all signals "aligned" w.r.t. unpipelined version (i.e., an interleaver doesn't reduce the delay of a circuit blow); (b) you need to interleave to get the best performance/cost (though no interleaving is not far); (c) multiple inputs should be handled correctly when interleaving; (d) if not treating the interleavers as also latching the inputs, then need a latch for each input signal on each path; (e) interleaving 3 ways is a bit tricky (and exceeds budget if done correctly); and (f) being consistent in assumptions and solution is important even if something in the assumptions is off.



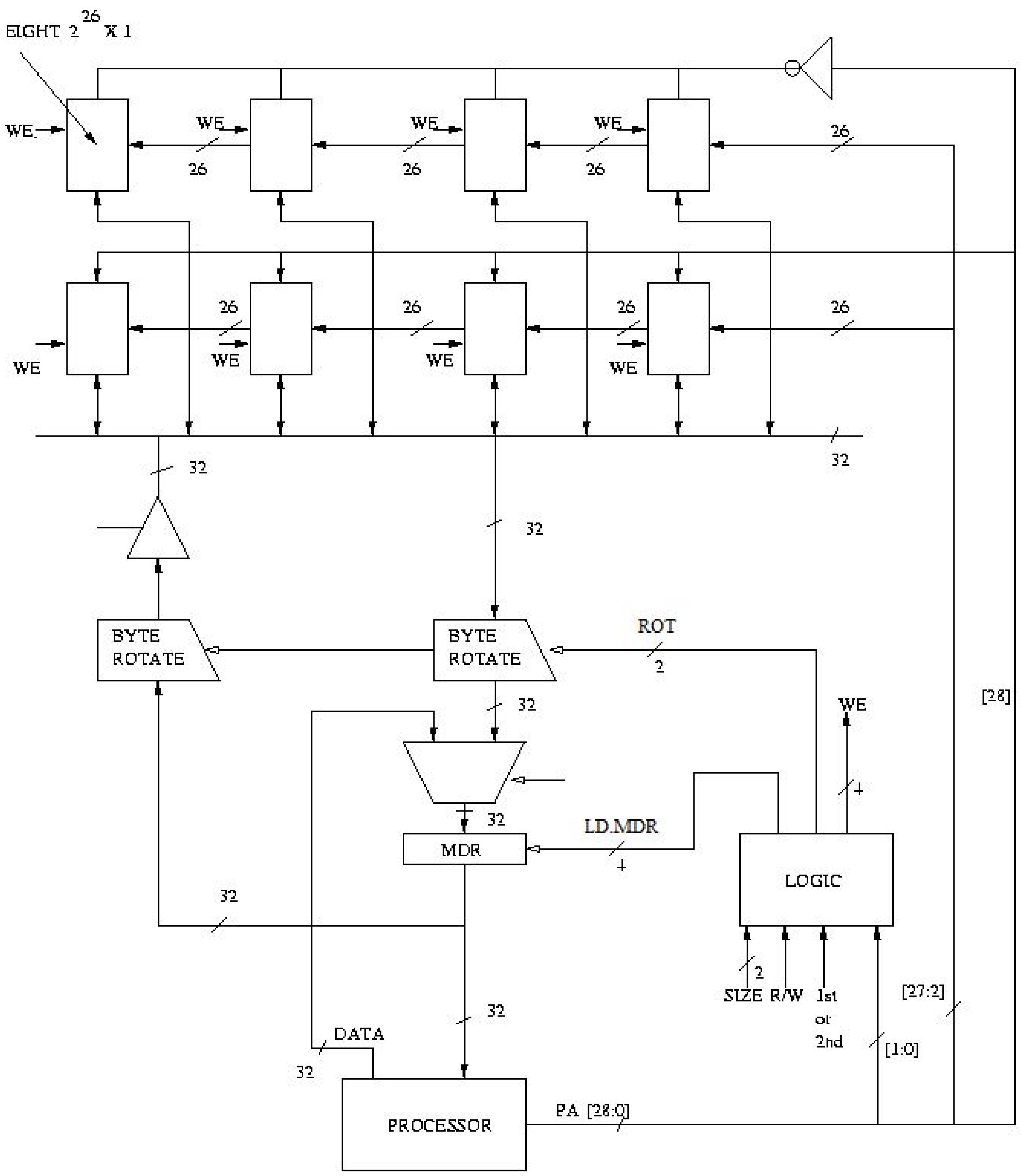
3. The figure on the next page illustrates the logic and memory for 512 MiB (byte addressable) of physical memory, supporting unaligned accesses. The ISA contains LDByte, LDHalfWord, LDWord, STByte, STHalfWord, and STWord instructions, where a word is 32 bits. Bit 28 serves as a chip enable (active high). If this bit is high the data of the memory is loaded on the bus, otherwise the output of the memory chip floats (tri-stated). All memory chips are accessed with a single address at a time and the processor issues two consecutive addresses when necessary to enable unaligned memory operations (similar to our discussion in class for unaligned hardware within the memory controller). The processor provides appropriate physical addresses on the 1st and 2nd access. Both of the rotators are right rotators. Construct the truth table to implement the LOGIC block, having inputs SIZE, R/W, 1st or 2nd access, PHYS_ADDR[1:0,] and the outputs shown in the figure below. Assume that the value of SIZE can be Byte (00), HalfWord (01), and Word (10). LD.MDR acts as a byte-granularity mask for the MDR (i.e. LD.MDR is a 4 bit signal. When LD.MDR[0]==1, it means lower 8 bits of MDR is filled with lower 8 bits of the 32 bit input (MDR[7:0] <- MDR_Input[7:0]). When LD.MDR[0]==0, the value of MDR[7:0] is not changed. Similarly, when LD.MDR[1]==1, MDR[15:8] <- MDR_Input[15:8], and MDR[15:8] is unchanged when LD.MDR[1]==0. Bits 2 and 3 of LD.MDR work the same for the other two bytes of MDR[15:8]. Clearly explain what function each output serves.

Table shown in the next page.
For aligned memory accesses, there is only 1 access, so the 1st/2nd input is not relevant; these cases are marked here with an “X” in the 1st/2nd input. In reads, the MDR has to be loaded with the data when it comes back, so we must set the proper bits in the byte mask; in writes, the MDR is loaded by the processor already and we do not want these values to change, so clear the proper bits in the byte mask. Notice that the values in the MDR are rotated into the expected positions from the *processor's* perspective, so we need to rotate between the MDR and the memory.
LD.MDR[3:0]: If 1, load data from processor or memory into MDR. This signal is 4 bits wide, because we can load each byte in the 32 bits independently. For writes, the data is loaded in the MDR before the memory access starts, so it should be 0 for writes.
ROT[1:0]: Right byte rotate the data from MDR to the memory and the data from memory to MDR. We can rotate 0-3 bytes right, so there are 2 bits for this signal.
WE[3:0]: Write enable for each chip. The memory is arranged as 1 byte wide per chip for a total memory bus width of 32 bits, so we have 4 bits in this signal, 1 for each byte.

PA[1:0]	**SIZE**	**RD/WR**	**1st/2nd**	**LD.MDR[3:0]**	**ROT[1:0]**	**WE[3:0]**
00	B	RD	X	XXX1	00	0000
00	B	WR	X	XXX0	00	0001
00	H	RD	X	XX11	00	0000
00	H	WR	X	XX00	00	0011
00	W	RD	X	1111	00	0000
00	W	WR	X	0000	00	1111
01	B	RD	X	XXX1	01	0000
01	B	WR	X	XXX0	11	0010
01	H	RD	X	XX11	01	0000
01	H	WR	X	XX00	11	0110
01	W	RD	1st	X111	01	0000
01	W	RD	2nd	1000	01	0000
01	W	WR	1st	0000	11	1110
01	W	WR	2nd	0000	11	0001
10	B	RD	X	XXX1	10	0000
10	B	WR	X	XXX0	10	0100
10	H	RD	X	XX11	10	0000
10	H	WR	X	XX00	10	1100
10	W	RD	1st	XX11	10	0000
10	W	RD	2nd	1100	10	0000
10	W	WR	1st	0000	10	1100
10	W	WR	2nd	0000	10	0011
11	B	RD	X	XXX1	11	0000
11	B	WR	X	XXX0	01	1000
11	H	RD	1st	XXX1	11	0000
11	H	RD	2nd	XX10	11	0000
11	H	WR	1st	XX00	01	1000
11	H	WR	2nd	XX00	01	0001
11	W	RD	1st	XXX1	11	0000
11	W	RD	2nd	1110	11	0000
11	W	WR	1st	0000	01	1000
11	W	WR	2nd	0000	01	0111

Legend

B(yte)	00
H(alf word)	01
W(ord)	10
RD(read)	0
WR(write)	1
1st	0
2nd	1



EIGHT 2^{26} X 1
WE
WE
WE
WE
26
26
26
26
26
26
26
26
WE
WE
WE
WE
32
32
32
BYTE ROTATE
BYTE ROTATE
ROT
2
32
WE
4
[28]
32
LD.MDR
4
MDR
LOGIC
32
2
SIZE
R/W
1st or 2nd
[27:2]
[1:0]
32
DATA
32
PROCESSOR
PA [28:0]